

Extensive Conformance Testing and Validation of FHIR® Data Exchange Variabilities

Abderrazek Boufahja¹ ^a and Tanmay Verma² ^b

¹GE HealthCare, Strasbourg, France

²GE HealthCare, Bellevue, Washington, U.S.A.

{abderrazek.boufahja, tanmay.verma}@gehealthcare.com


Keywords: HL7®, FHIR®, Conformance, Validation, Testing, API.


Abstract: The emergence of FHIR® standard during the last years was accompanied with the development of many FHIR® servers, some of them are commercials, and many are open-source projects, with a wide deployment in production. The FHIR® standard defines a complete RESTful API allowing access and sharing of clinical resources participating in dozens of healthcare workflows. The defined API comes with a complete list of variations in CRUD operations and in search queries. For instance, every search parameter comes with multiple searching flavours, making the implementation of the hundreds of search parameters complex, and the servers capability claims hard to verify by FHIR® clients, especially for those who use edge search capabilities. In this paper, we used a method to test exhaustively the large number of variabilities in the RESTful FHIR® API that can be implemented by a FHIR® server, by generating thousands of test scripts, using directly the formal description of the FHIR® standard. The method allows validating the different search variabilities and brings a deep view of the capabilities of the tested FHIR® servers. An implementation of the method was experimented, and the generated scripts were tested with multiple FHIR® servers. The testing of different FHIR® servers highlighted the conformance of most of them to the FHIR® standard, even if some discrepancies between the claims of some FHIR® servers and their current implementations were observed and analysed. We concluded the paper with an analysis of the search variabilities with commonly found behaviours and limitations. The overall work highlights the importance of a complete and strong testing strategy for a better integration and patient care.

1 INTRODUCTION

Fast Healthcare Interoperability Resources (FHIR®) (Benson and Grieve, 2016; Ayaz et al., 2021) standard is widely adopted today across the globe as the new healthcare standard to exchange clinical data (Braunstein, 2022). It defines clinical concepts as resources, and defines an API to create, manipulate and search these resources. Dozens of resources were defined, and hundreds of search parameters can be used to query the FHIR® API (HL7, 2023). Also, for every search parameter, many variabilities may exist, based on the type of the search parameter. Thus, FHIR® API counts thousands of variabilities and query possibilities. Due to this complexity and the richness of the API variabilities, the FHIR® servers implementing FHIR® standard as a façade for FHIR® resources

expose usually different supported variabilities, and usually target implementation of the needed capabilities for the use cases they are supporting and targeting. Many FHIR® servers come as a full FHIR® server implementation as well, meant to be used as a repository of FHIR® resources. These FHIR® servers implement many capabilities of the FHIR® API, and try to cover as much as possible, to cover the most possible use cases, and to stay agnostic to the use cases variations. FHIR® server adopters can only rely on the declared capability statement performed by FHIR® servers providers. Many tools allow testing the capabilities of FHIR® servers, but many of them are only testing the high-level capabilities, without going deeper on the test variations. In this paper, we describe a method to perform extensive conformance testing of FHIR® API, and reverse generation and validation of the FHIR® server capability statement.

^a  <https://orcid.org/0000-0002-6481-2185>

^b  <https://orcid.org/0009-0000-2337-7239>

2 STATE OF THE ART

2.1 FHIR® Standard

Fast Healthcare Interoperability Resources (FHIR®) (Benson and Grieve, 2016) is a new generation standard developed by HL7® (Health Level Seven) organization, to respond to the technology evolution of the last years. The first version of the standard was proposed in 2011. Since then, multiple versions were released. The latest version available during this work is the version R5 (5.0.0). FHIR® defines two principal components: resources and APIs. Resource types define the data elements, constraints, and relationships with other resource types. They describe a modelling of real-world clinical information. There are nearly 150 resource types defined by FHIR® covering most of the known entities used in healthcare. FHIR® standard keeps a large flexibility to cover healthcare use cases. It follows the Reuse and Composability principle, following the rule 80/20: FHIR® standard makes the modelling of the most known elements, and keeps the possibility for specific use cases to extend the core resource types. FHIR® defines APIs to access the resources, and to manipulate these resources. The FHIR® API is following Level 2 of REST Maturity Model (Ozdemir, 2020). The APIs defined by FHIR® have three levels of interactions: APIs at the resource instance level, APIs at the resource type level, and APIs at the system interaction level. Instance level interactions allow mainly CRUD operations, and compartment access for defined HL7® compartments. Resource type level allows mainly search operations, and the system level APIs allow batch and transaction queries, as well as few other capabilities. The most complicated API capability is the search at resource type level, as every resource has sometimes dozens of search parameters, and every search parameter can have many searching variabilities.

FHIR® REST API is described under the page RESTful API from the FHIR® standard, and the search variabilities are described in the “Search” page from the FHIR® standard (HL7, 2023). The FHIR® API variabilities are usually described under a CapabilityStatement resource, exposed using the capabilities interaction. A CapabilityStatement can describe a FHIR® client or a server of resources. In our study, we focus on server capabilities. The CapabilityStatement resource has three flavours: instance, capability, and requirements. CapabilityStatement can describe different behaviours of a FHIR® server, like the security of implementation, the implemented resources, the system level interaction, the supported resources, the operations for every resource, and the supported

search parameters for every resource.

2.2 Search Parameters Variabilities

Search parameters variabilities are described in the “Search” page in the FHIR® standard. A search parameter can be described in a minimal manner in the CapabilityStatement, with just exposing the search parameter name, or it can be a complete description of the search parameter variations with a link to a SearchParameter resource, describing the supported search variabilities. Every search parameter can have multiple variabilities, many of them can be documented in the SearchParameter resource. For instance, modifiers can be described as part of the SearchParameter resource, comparators as well. Comparators are also called prefixes. All the variabilities related to modifiers and comparators are described in the documentation “RESTful API Search” under FHIR® standard, and most of them depend on the type of the search parameter. FHIR® defines nine different search parameter types: date, number, quantity, reference, string, token, uri, composite, and special. Every search parameter type has many variations, like comparators, modifiers, the structure of the searched value, and its precision.

Every resource has a specific number of search parameters, defined in the FHIR® standard (HL7, 2023). Every resource of type DomainResource, can have an extra search parameter: ‘_text’. In FHIR® R4, all the resources extend DomainResource, except Bundle, Parameters, and Binary. However, all the resources extend Resource type, directly or through inheritance. Thus, all the FHIR® R4 resources can implement any search parameters from the Base Resource definition: ‘_content’, ‘_id’, ‘_lastUpdated’, ‘_profile’, ‘_query’, ‘_security’, ‘_source’, and ‘_tag’.

Figure 1 describes the number of search parameters for FHIR® R2, R3, R4 and R5. We collected the number of the search parameters that FHIR® resources can support, but we excluded the common search parameters described above, which are inherited from DomainResource and Resource. The number of resources did rise between R2 and R5 release and was slightly stable after the release R4. However, the number of search parameters continues to rise every release. For R4, there are more than 1600 search parameters.

Some common search parameters are quite complicated and can have dozens of variations. For example, the search parameter ‘_has’ is described by the notion of reverse chaining. Every other resource that can reference the current resource through a search parameter can be used to search on the current in-

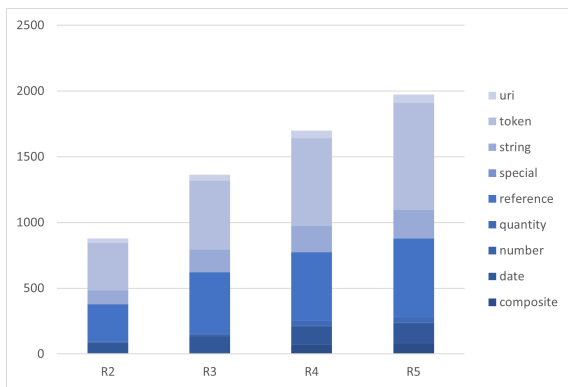


Figure 1: Number of search parameters per FHIR[®] version, excluding common search parameters.

stance of the resource. ‘_has’ search parameter can also be nested, and in this case, we can have complex search parameter structures.

For a better understanding of the FHIR[®] API variations, we worked on a quantitative identification of all the variations. For every resource, for every search parameter, we identified the possible variations that can be filled by a FHIR[®] server API. For every search parameter, based on its type, we identified the different variations related to that search type. For search parameters of type reference, we identified the chained search parameters as well. For some search parameter types, we did go beyond the definition and variations defined in the SearchParameter resource. For example, for string-based search parameters, equality to a value was refined to three variations: strict equality, equality with case sensitive variation, and equality with a begin-with behaviour. In FHIR[®] standard, string matching without modifiers returns results that have insensitive matching elements equal or start with the searched value; however, FHIR[®] servers may implement one or the other variability, on purpose or by error. For date-based search parameters, even if the FHIR[®] standard describes that any precision level can be used in searching values, some FHIR[®] servers may not support all the variabilities in searching by year, month, day, minutes, or seconds. Making these capabilities as a unit of variability within a FHIR[®] server allows having a deep understanding of the capabilities of the FHIR[®] server, which goes beyond what FHIR[®] defines. We identified for instance more than 50 variabilities for every date-based search parameter. Every precision can be coupled with the different prefixes used on dates, and every combination of date precision and prefix is a searching variation. For token-based search parameters, the structure of the search token can be (code), (system|code), (|code), or (system|). Also, it can have multiple modifiers like :text, :not, :above,

:below, :in and :not-in. Besides chained parameters, reference-based search parameters can have three different searching structures: searching by (id), by (ResourceType/id), or searching by the full URL. Also, it can support modifiers like :identifier, or an explicit resource type as a modifier.

More than 3500000 variabilities were identified and can be supported by FHIR[®] R4 servers. This number does not include iterating on the parameters ‘_has’, ‘_include’, and ‘_reinclude’. It includes only the first level for chained parameters, and it does not include combination of search parameters. This estimation does not include Compartment-based variabilities either. ‘_has’ variabilities represent 80% of the overall variabilities, and more than 98% are related to the parameters ‘_has’, ‘_reinclude’ and chained parameters. The remaining is nearly 60000 variabilities, which is still a substantial number to test and to cover for a FHIR[®] server.

2.3 CRUD Operations Variabilities

FHIR[®] API supports level 2 of the REST Maturity Model (Ozdemir, 2020; Webber, 2010). In FHIR[®] R4, interactions with the FHIR[®] servers are divided in three categories: six instance level interactions, three type level interactions, and four system level interactions. Every interaction has multiple variabilities. For example, ‘update’ interaction in instance level can be described with two variabilities: update of an existing resource, and update-as-create behaviour. ‘patch’ interaction can be refined to 15 different variations. HL7[®] defined two major methods to do patching: JSON patch and FHIR[®] patch, and every method has many variations: adding, copying, moving, etc. Also, conditional operations make every CRUD operation refined to multiple variations and use cases, based on the used condition and its applicability. We identified 27 variations for CRUD operations on every FHIR[®] resource, which represent nearly 4000 variations.

2.4 Validation Methods and Tools

Multiple tools and methods exist as FHIR[®] validation tools, many of them are open-source. FHIR[®] standard and community tried to document most of these resources in the FHIR[®] standard itself through the Implementation Support module and the Validation Resources page, or in related HL7[®] confluence pages (HL7, 2024a; HL7, 2024b). FHIR[®] testing tools can be divided in two categories: resources validation tools and API exchange testing tools.

2.4.1 Resources Validation Tools

Resources validation principles and methods are described in the FHIR® standard through the page Validating Resources. There are mainly six methods checking eight different aspects. FHIR® standard comes with an XSD schema and schematrons, which can be used by many XSD-based and schematrons-based validation tools. Also, FHIR® standard provides a JSON schema, which can be used with many JSON validation software. FHIR® community maintains a FHIR® validation tool that can be used in command line to validate FHIR® resources against standard FHIR® requirements, or against extended requirements related to FHIR® profiles. FHIR® community provides as well a web-based validation tool that can be used online to validate FHIR® resources (Otasek, 2024). Some studies were performed as well in order to validate FHIR® RDF presentation of resources using Shape Expressions (ShEx) (Solbrig et al., 2017). This ShEx based method can be used as well to validate XML and JSON presentations of FHIR® resources through transformation to RDF presentation. Many provide open-source or online FHIR® content validation tools. Here is a non-exhaustive and unordered list of online content validation tools: Simplifier validation tool, Firely validation tool, Inferno validation service, Gazelle FHIR® validation tool, Infoway FHIR® Validator, Aidbox FHIR® Schema Validator and clinFHIR resource validator. Some FHIR® sandboxes provide as well \$validate operation like Aegis sandbox or HAPI FHIR® online server. We can note as well FHIR® Notepad++ Plugin and FHIR® tools plugin for Visual Studio Code (Lagger, 2023), which provide a validation capability for FHIR® resources.

The existence of all these tools confirms the maturity of FHIR® content validation tools, and their wide usage amongst FHIR-based products providers.

2.4.2 FHIR® API Exchange Testing Tools

FHIR® resources are exchanged usually through the REST API defined and described in the FHIR® standard, even if other exchange mechanisms could be used. Most of the FHIR® servers provide testing sandboxes to accelerate integration with FHIR® applications. For instance, most of the EMRs provide some testing sandboxes with some connectivity validation process and are particularly useful to reach a high level of interoperability for the FHIR® clients. Some open-source FHIR® servers can be configured and installed in a private network and used to test FHIR® clients' implementations against these FHIR® servers. Here is a non-exhaustive and un-

ordered list of open-source FHIR® servers: HAPI FHIR® server (Hussain et al., 2018), LinuxForHealth FHIR® Server (Opie, 2024), Microsoft FHIR® server (Opie, 2024), FHIR® Candle (Canessa, 2024), Pascal FHIR® Server (HealthIntersections, 2024), and Spark (Kramer, 2024).

FHIR® standard defined TestScript resource, providing agnostic and interoperable method to share test designs and test definitions, in executable format, with computable actions and interpretable verification instructions. Also, FHIR® defines TestReport resource which can be used to share a summarized testing report following the execution of a TestScript. Other methods exist as well, defining generic structures for tests and test execution steps (GITB, 2015; Scanlon, 2024).

Many tools allow testing FHIR® resources exchange through the FHIR® API. Some of them are open-source, others are proprietary or having hybrid access. For instance, all HTTP REST testing tools and automation tools can be leveraged to test FHIR® servers APIs, like Postman or SOAPUI (SOAPUI, 2024). We can list this non-exhaustive and unordered open-source and commercial FHIR® APIs testing tools:

- Touchstone (Walonoski et al., 2018) developed by AEGIS, offers automated FHIR® testing for servers and clients implementations, leveraging FHIR® TestScript resource. It can be used to automate FHIR® data exchange testing, can be used with pre-designed TestScript resources, and can be used as a testing framework to add new test scripts.
- Crucible (Walonoski et al., 2018; Scanlon, 2024) developed by MITRE, offers a set of open-source testing tools for FHIR®. Can be used to test FHIR® servers data exchange conformance and can score patient records.
- Inferno (Kramer and Moesel, 2023) is an open-source tool that helps testing conformance to FHIR® standard. Besides its resources' validator, Inferno provides a web-based application to execute online testing using many available test kits. Every test kit is a list of defined tests that can be executed against a specific FHIR® server endpoint. FHIR® servers can use predefined test kits or can develop their own test kits using Inferno framework.
- Caristix Test (Caristix, 2021) provides the possibility to automate testing of FHIR® implementations using Scenario Editor.
- NIST FHIR® Toolkit (NIST, 2024) is an open-source FHIR® testing tool, mainly for IHE® MHD

FHIR[®] based profile.

- Gazelle PatientManager tool allows simulating some initiating actors as FHIR[®] client for some FHIR-based IHE[®] profiles, like PDQm or PIXm. Also, it can act as a FHIR[®] server to test some IHE[®] profiles.
- TestScript Engine (MITRE, 2023) is an open-source testing engine. It is able to interpret and execute TestScript FHIR[®] resources and generate TestReport resources following command line execution.

This list is not exhaustive, we think many other proprietary or open-source FHIR[®] testing tools and frameworks may exist. Some of the tools come with preconfigured test definitions, some tools provide a framework to create test definitions, which may follow TestScript resource structure, or some other proprietary structures.

The vast number of variabilities makes FHIR[®] servers claims difficult to check and to validate and makes sometimes the integration of clients with heterogeneous FHIR[®] servers complex without staging and testing phases. We used in this paper a method to generate unitary verification tests directly from the FHIR[®] standard, to cover exhaustively all the possible variations in the FHIR[®] API, and to provide a clear knowledge on the FHIR[®] server capabilities.

3 METHOD

To test agnostically all API variabilities that can be implemented by a FHIR[®] standard, a huge number of tests needs to be written, based on the variabilities analysis described above. The aim of this method is to avoid writing test definitions or test scripts, and to generate them automatically based on computable artifacts from the FHIR[®] standard. In fact, writing manually all the identified tests is time consuming, and error prone, which is something we are avoiding with our method.

From the FHIR[®] standard, two main artifacts are used: the Resource Type profile.json files containing the StructureDefinition of the FHIR[®] resource types, and the file search-parameters.json, containing a computable definition of all the search parameters in the FHIR[®] standard, as described in Figure 2. Also, the generator engine takes a list of test script templates as input, containing a template of test definition for every variation for every search parameter type. Every template is meant to be compiled using inputs related to tested resources and tested search parameters. The

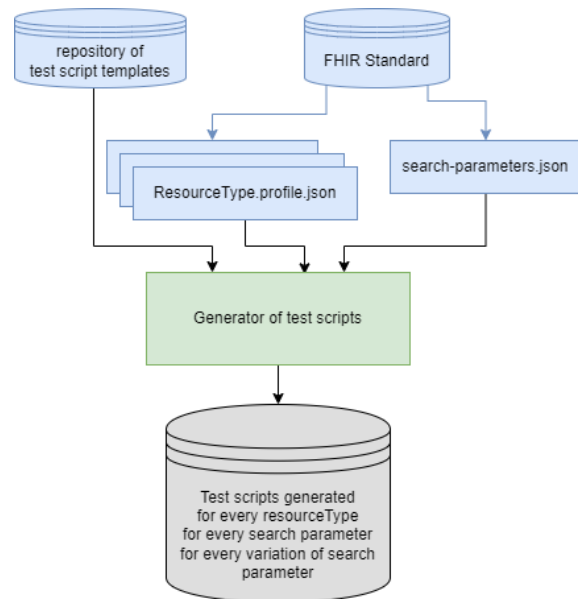


Figure 2: Test scripts generation overview.

generator engine uses these inputs as described in Figure 3.

The list of resource types is extracted from the FHIR[®] standard. For every resource type, the list of supported search parameters is extracted from the search-parameters.json file. Every search parameter is defined with some information, like the target resources, the type of the search parameter and the path in the FHIR[®] resources to the elements that need to be mapped to the search parameter. The path of the search parameter is used to validate the returned bundles following test scripts execution, it is used to compare the searched values, and the returned resources content. For every search parameter found in search-parameters.json file, the list of variations is identified based on the search parameter type. For every variability, we use the defined repository of test scripts templates to identify the test script template to be used for the specific identified variation. Then, based on that template, the resource StructureDefinition, and the search parameter attributes, the generator engine generates the final test script of the variability. For example, for the Organization resource, we have in search-parameters.json multiple identified related search parameters, like type, partof, and address. The parameter type is a token, for which we identified in our study 15 different possible variabilities. For every variation type, a test template is defined, and is used to generate a test script meant for testing only a specific variability on the ‘type’ search parameter for the Organization resource type. Every variability type for every search parameter type has its own test script template, which contains some testing steps and

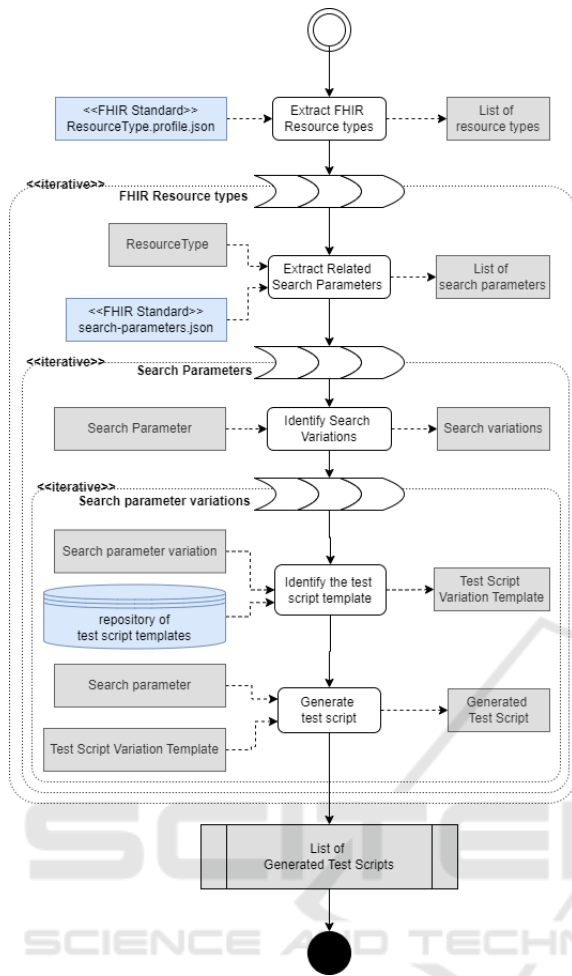


Figure 3: Test scripts generation process.

testing logic, including negative testing and validation mechanism of the returned Bundle following a search query. For instance, one of the authors has a patent application related to this method, with more technical details. The generation of the CRUD variabilities for every resource was following the same pattern as well, with test script templates used to generate test scripts for every resource type.

When executed on a FHIR® server, the generated test scripts allow identifying the capabilities of the tested FHIR® server through testing, instead of identifying these capabilities through only accessing the CapabilityStatement declaration. This enables some comparison and validation of the declared capability statement, as described in Figure 4.

The first step before executing the generated test scripts is the initialization of testing variables. In fact, to search FHIR® resources using search parameters, we need to identify existing resources in the FHIR® server – or create new FHIR® resources – and ex-

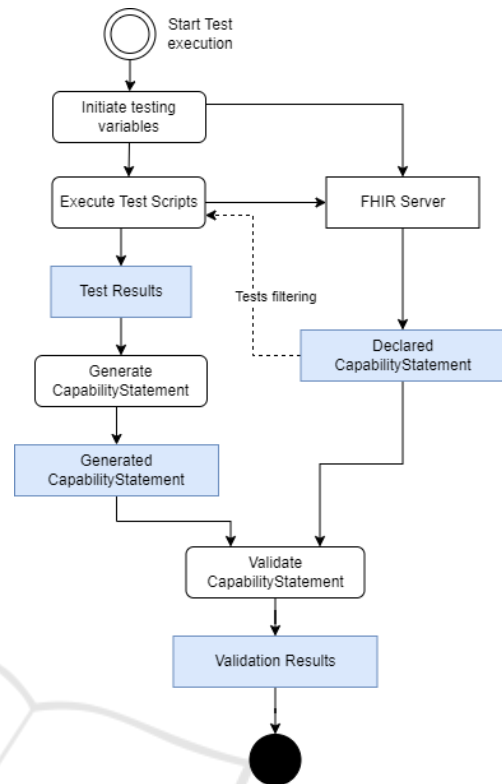


Figure 4: CapabilityStatement generation and validation.

tract the right values for all tested search parameters, using the path definition identified in the search-parameters.json file, from FHIR® standard. When the FHIR® server supports create interaction, the initialization of the FHIR® server is done with static FHIR® resources as part of the initialization process.

During the execution of the generated test scripts on a FHIR® server API, test results allow detecting the different searching variabilities. These variabilities results are used to generate a computed CapabilityStatement which can be compared with the declared CapabilityStatement from the FHIR® server. This comparison allows us to highlight the differences between what is claimed, and what is implemented. Because of the considerable number of generated test scripts, the declared CapabilityStatement by the FHIR® server can be used to filter the tests to be executed.

4 APPLICATION

4.1 API Testing with Inferno

We implemented the method described above using Inferno framework (Scanlon, 2024), to generate test

scripts for API variabilities for FHIR® R4 standard. Inferno has a method to develop test kits, and the performed implementation was through generating test files, cascaded together to map every variability to a generated test, executed using Inferno UI or its API. Generation of tests related to the variabilities on ‘_has’, ‘_reinclude’ and chained parameters were ignored in our testing. Over the remaining 60000 variabilities, our templates and our generation of tests resulted in covering more than 42000 variabilities. The following types of search parameters were covered completely in our testing: string, date, number and uri. ‘reference’ type was not fully covered as chained parameters, and ‘identifier’ suffix were complicated to test, and needed further parsing and following of references on the collected resources. Token-based search parameters were nearly fully tested with all variabilities, except subsumes on ‘above’ and ‘below’ suffixes, as well as ‘in’ and ‘not-in’ suffixes. This is because of the complexity of verifying these operations, which needs connectivity to a terminology service. Composite based search parameters were not tested, but their number is quite small compared to the other types. Common search parameters were tested in all the resource types, with all their variations. The tested common parameters are _id, _lastUpdated, _tag, _profile, _security, and _source. Most CRUD operations and variabilities were tested as well. Compartment based resources APIs were not tested during this application.

Test data were defined to fill the tested FHIR® server with initial FHIR® resources. The definition of the test data was one of the major steps, as these data needs to be heterogeneous and rich enough to create an optimal testing environment. These test data were designed to enable testing of most of the targeted search parameters. For every executed test, four statuses can be reported:

- Pass: the variability is supported.
- Fail: the test was failing, the FHIR® API does not support the tested variability.
- Crash: the test was crashing during execution.
- Missing test data: the collected test data are not sufficient to test the targeted variability.
- No test defined: the variability cannot be tested nor verified.

Failed tests are usually because the asserts and verifications as part of the tests were failing. When a test crashes, it is mostly because the variability is not supported. For example, the API may return a 400 Bad Request response. Tests executed and marked as ‘missing test data’ are usually because of missing searching values. Before executing searching on

parameters of a specific resource, we need to collect FHIR® resources from the server. The collected FHIR® resources may not contain all the needed information that allows us to perform all the search parameters. Inferno framework enables for instance parameters sharing between tests, which enable search parameters initialization through create and search phases. The tests marked as ‘missing test data’ or ‘no test defined’ cannot confirm or deny the implementation of a specific variability of the FHIR® server API.

4.2 Extensive Testing Results

We executed the generated tests on many open-source FHIR® servers, as well as on some available sandboxes, many of them are described in the state of the art. We anonymized the FHIR® server names, the goal of the analysis is not to compare between them, but to identify implementation variations and common behaviours. The overall number of executed tests was nearly 400 000 tests.

4.2.1 Overall Testing Results

Testing the FHIR® servers was time challenging, as the execution of more than 42000 tests per server takes several hours, and sometimes days, to finish execution. Every test makes many FHIR® queries to test combinations of positive and negative test steps. The testing results are resumed in Figure 5 and Figure 6. During testing, we did not filter based on the declared CapabilityStatement, we executed the full test suite.

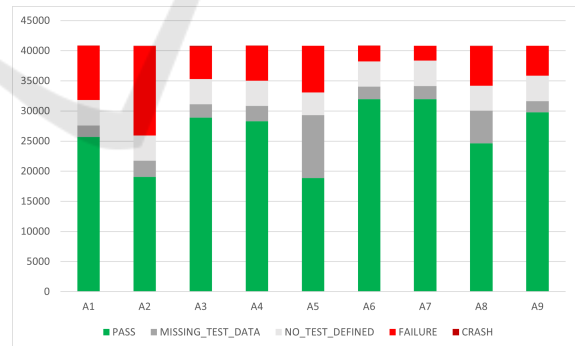


Figure 5: FHIR® servers search variabilities coverage.

Different open-source FHIR® servers support wide amount of configuration parameters, which may activate searching and operations capabilities, like indexing of the missing elements, activation of text search capabilities, activation of update-as-create capability, acceptance of non-resolvable references in FHIR® resources, and many other configurations. During the testing and configuration of the tested open-source FHIR® servers, we tried to activate the

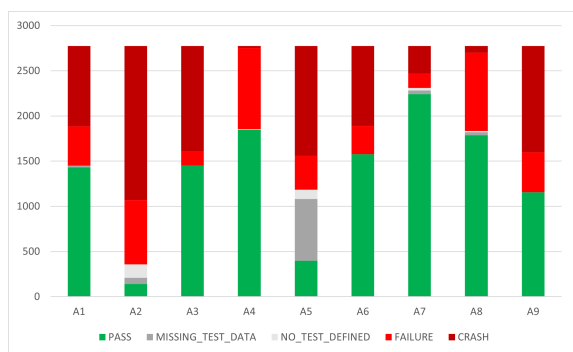


Figure 6: FHIR® servers CRUD variabilities coverage.

most possible and understandable level of indexation and capabilities, but we acknowledge that some capabilities may be missed during the configuration process.

Most of the tested FHIR® servers presented a good coverage of the different tested search capabilities. However, we remarked that some FHIR® servers were presenting many testing failures, highlighting non-supported features by their APIs. In CRUD variabilities testing, many tests were crashing, especially because the received responses were with an HTTP status indicating an error, highlighting non-supported capabilities.

4.2.2 Search Variabilities Analysis

For variabilities related to search parameters, we grouped the different tested results per variability type and per resource type. The Figure 7 describes the number of failures per variability type, tagged by the types of the search parameters. The Figure 7 analyses variability types where we have more than 200 encountered failures, during testing with the FHIR® servers described above. The included tests are only related to supported search parameters by tested servers. Every line in Figure 7 describes a tested variability, like for example searching by some specific prefix for date-based search parameter, or searching by ‘exact’ modifier for a string-based search parameter. We grouped all the tests here per variability type.

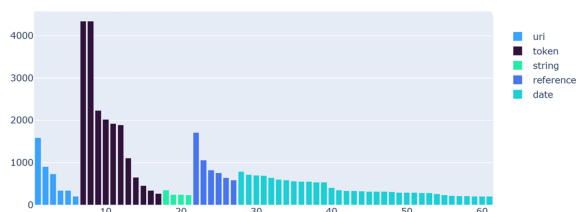


Figure 7: Number of failures per variability type.

We remark that all the tested search parameter types have some high number of failures, except

quantity and number-based search parameters, which is understandable as the number of the related search parameters is already low in FHIR® standard.

The less supported variabilities are related to token-based search parameters and are related to searching by the modifiers ‘above’, ‘below’, ‘text’, ‘missing’, ‘not’ and ‘of-type’. The performed tests highlight elevated level of failure on these modifiers, which means a totally not supported feature in some FHIR® servers, or a mis-implementation of the variability. Searching for equality to (system|) variability type was failing, which concluded the low support and implementation of this variability in token-based search parameters.

Searching with the modifiers ‘above’ and ‘below’ were the most encountered failing tests for uri-based search parameters. For reference-based search parameters, searching by the full URL of the reference was poorly supported. An interesting, detected failure was related to searching references by resource ID. It seems some FHIR® servers prefer searching by Resource/ID structure instead of searching directly by the resource ID.

Many date-based search variabilities were failing as well. Searching with ‘ap’ prefix was often failing (> 3000 tests failed), highlighting poor support of this prefix. Testing with the prefix ‘ne’ was often failing (~ 2500 tests failed). Variabilities related to searching with minutes precision were often failing (> 2500 tests failed). More than 1000 tests were failing related to search prefixes ‘le’ and ‘ge’, which is an alarming observation: FHIR® clients commonly use these prefixes, non-compliance or misinterpretation of their meaning can have harmful impact on patient care.

We remark that string-based search parameters were well implemented for instance in most of the tested search parameters. Number and quantity-based search parameters were not shown in Figure 7, as their number was small. To have a better representation, Figure 8 describes the rate of failure over the total number of executions of a specific variability type for a specific search parameter type. The included tests are only related to supported search parameters by tested servers. The diagram contains only failures with a rate higher than 0.1 of failure occurrence.

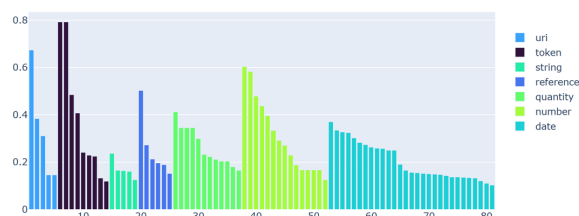


Figure 8: Rate of failures per number of tests executed for each variability type.

Token-based search parameters still appear in Figure 8 with an important level of failures, related to the same variabilities highlighted above: modifiers ‘above’, ‘below’, ‘text’, ‘missing’, ‘not’ and ‘of-type’. Uri-based search parameters were also failing tests related to variabilities ‘above’ and ‘below’, which in both cases describes a poor support of these variabilities. Date-based search parameters and reference-based search parameters were as well present with high number of failure percentage, still related to the same variabilities described above.

Although the number of tests related to number-based search parameters is low, the percentage of failure on the different searching variabilities is high. Most of the failed tests are related to the usage of prefixes, which was poorly supported or implemented, especially the prefixes ‘ap’, ‘eb’, and ‘sa’. The same observation can be concluded on quantity-based search parameters, with a smaller number of identified failures. The prefixes ‘ap’, ‘eb’ and ‘sa’ can be considered for instance edge cases.

5 DISCUSSIONS

SearchParameter resource covers many variations for all search parameters and can be linked to the CapabilityStatement of the FHIR® server to provide more details about the level of implementation. Even so, this may not be sufficient to describe all the possible implementation variabilities, which may not be conformant to the definition and requirements defined by FHIR®. For example, searching on string parameters is not case sensitive, but some FHIR® API implementations may implement only a case sensitive searching parameter, accidentally or intentionally. This kind of variability may be difficult to describe in the SearchParameter resource. Another example will be date-based search parameters. FHIR® server implementations may implement searching by year, month, date, minutes, second, or fraction of seconds capabilities, and for every precision type, the implementation can support the different prefixes that the standard defined (‘eq’, ‘gt’, ‘lt’, etc.). The number of variations for date-based search parameters was more than 50, beside some common variations to all search parameters, like support for ‘missing’ modifier and OR/AND searching variabilities.

We described a method that allows generating automatically all the tests for all the variabilities of the FHIR® API, using the formal definition of the FHIR® standard. The generated tests allow us to understand the capabilities of the FHIR® API, and to compare and validate the claimed capabilities within the FHIR®

CapabilityStatement declared by the tested FHIR® server. The method described in this study can be extended to test as well custom resources in FHIR® servers and their search variabilities (Boufahja et al., 2021). We implemented this method by writing templates to generate tests as part of a test kit integrated locally with Inferno testing platform. Some tests were not generated as part of the application due to the complexity of the tests. We executed the generated tests against many testing servers and sandboxes, including open-source FHIR® servers. The goal was not to compare FHIR® servers’ providers, but to enhance generic knowledge on servers’ capabilities and common issues and discrepancies. The implementation of the method was using Inferno testing framework, which was very efficient for implementing and executing the different tests.

Many difficulties were observed during testing date-based search parameters. First, the search parameters of type date can refer to multiple data types, like dateTime, Period, and Timing, which makes the verification of the search operation complex. In date comparison, there are some complexities as well in handling the time zone. For date search that goes to the time level, we always include the time zone as part of the search parameter, to avoid server-based interpretation of the meaning of the query. For many of the tested FHIR® servers, many tests were failing with le, ge, lt and gt prefixes, which was a surprising finding. This can be explained by the complexity of the interpretation of these prefixes and their temporal meaning, for which we experienced this complexity during writing the verification code for the executed queries. For instance, some of the found errors can become problematic for the patient care, when the date searching queries are returning wrong information, or missing information, and this enhances the importance of clearly identifying the claimed variabilities by FHIR® server and having a strong testing process for all the claimed variabilities. Testing ‘ap’ prefix was complicated as well and hard to verify, especially because of the lenient interpretation of this prefix, which differs between implementers. Quantities and numbers’ precisions were not tested. A generic relative period was considered during the verification of the searching results related to quantities and numbers.

We performed as well testing combinations of AND and OR values for all search parameters. The performed tests confirmed that servers may implement these variations for only a subset of the searching parameters. FHIR® clients should be vigilant regarding these kinds of variations within the same FHIR® server.

Common search parameters were tested for every

resource. We remarked a strong adherence of FHIR[®] servers to support these common search parameters. However, some edge case variations were failing in some servers, like testing `_id:missing`, `_id:not` and `_lastUpdated:missing` variations.

Many FHIR[®] servers and sandboxes sometimes support only combined or mixed search parameters. Because of these kinds of requirements, results from testing with sandboxes need to be handled carefully, as failure of executed tests may not be because the tested server is not supporting the tested variabilities, but because of mandatory mixed search parameters.

Most of the tested FHIR[®] servers were not fully supporting all CRUD operations variabilities, and many tests were failing. For example, following a DELETE operation, some FHIR[®] servers were responding with 404 status code for a deleted resource, instead of 410 status code. For update-as-create operation, some FHIR[®] servers were responding with 200 status code, instead of 201.

Since the search parameters are generated based on an initial collection and initialization of the FHIR[®] resources, returned Bundles can be different between the different tested servers. Even within the same sandbox, executing the generated tests twice may result in different FHIR[®] queries, due to the variations on the content of the FHIR[®] servers and their hosted resources. Because of this behaviour, sometimes tests can pass because the tested data were optimal and allowed to have consistent queries and results, even if we enforced negative testing and results verification in all the generated test scripts. Thus, we consider that the passing tests are only a strong indication that the searching variabilities may be well implemented.

Even if all the executed tests were generated automatically based on the FHIR[®] standard, and that we verified the relevance and the well design of the templates, edge cases may not be fully tested. The failed tests were not all verified manually to check the relevance of the encountered failure, even if dozens of manual verifications of failed tests were performed and confirmed.

The comparison between the test results and the claimed capabilities of the FHIR[®] servers highlighted discrepancies. For instance, some FHIR[®] servers claim support for patch operations, without highlighting which variabilities are supported. Testing their APIs highlighted only a partial support. Also, in some FHIR[®] servers, we find that they are supporting some extra search parameters that they are not declaring in their FHIR[®] conformance statement.

This extensive testing method and application highlighted the importance of a strong testing strategy and mechanism for all the possible implemented

variabilities, which should start first by documenting all the different supported variabilities. Weak testing mechanism leads to API interpretation errors and can become dangerous for the patient care in some cases.

6 CONCLUSIONS

The FHIR[®] standard defines a complete API for data access and search, with nearly 150 resources and more than 1500 search parameters combinations. Every search parameter is defined with many variabilities and usage. We identified for instance more than 3500000 possible variations in the FHIR[®] API, 98% of them are related to only three kinds of search variations. Such variability highlighted the complexity of putting in place a complete campaign for testing all the API variations. We described in this paper a method to generate test scripts using the formal description of the FHIR[®] standard in order to cover all possible variabilities. The method uses test templates defined based on the search variabilities analysis. The report from executing the generated test scripts can be used to validate the FHIR[®] server claims declared in their CapabilityStatement. We applied the method using Inferno framework through implementing and generating a subsequent number of the identified test scripts variabilities. The generated test scripts were executed with many available FHIR[®] servers. The results of execution showed many differences in the implementation and coverage of the tested FHIR[®] servers. Many variabilities were poorly supported, highlighting that FHIR[®] clients should be aware of such limitations in FHIR[®] servers, during developing FHIR[®] based applications. Some claimed capabilities in FHIR[®] servers were not fully supported, and the execution of the generated tests allowed to identify clearly every supported and non-supported variation. Different commonly used variabilities were failing occasionally, like searching on date with some common prefixes. Such testing failures highlight the importance of having a complete testing suite and a testing strategy for FHIR[®] servers, to provide a better integration experience and a better patient care.

ACKNOWLEDGEMENTS

We acknowledge strong GE HealthCare support during this study from Science and Technology Organization personnel for their feedback and support that helped in formulating the conclusions.

REFERENCES

- Ayaz, M., Pasha, M. F., Alzahrani, M. Y., Budiarto, R., and Stiawan, D. (2021). The Fast Health Interoperability Resources (FHIR) Standard: Systematic Literature Review of Implementations, Applications, Challenges and Opportunities. *JMIR Medical Informatics*, 9(7).
- Benson, T. and Grieve, G. (2016). *Principles of Health Interoperability: SNOMED CT, HL7 and FHIR*. Health Information Technology Standards. Springer International Publishing, Cham.
- Boufahja, A., Nichols, S., and Pangon, V. (2021). Custom FHIR Resources Definition of Detailed Radiation Information for Dose Management Systems. In *Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies*. SCITEPRESS - Science and Technology Publications.
- Braunstein, M. (2022). *Health Informatics on FHIR: How HL7's API is Transforming Healthcare*. Springer.
- Canessa, G. (2024). fhir-candle When you need a small FHIR. Open-Source Tooling, Webinar Series.
- Caristix (2021). Caristix Test Scenario Editor. <https://caristix.com/tutorials/testing/fhir-server-capabilities/>. Accessed: 2024-09-05.
- GITB (2015). CEN Workshop Agreement Global eBusiness Interoperability Test Beds GITB Phase 3 | Joinup. Technical report.
- HealthIntersections (2024). Pascal FHIR Server Reference Implementation. <https://github.com/HealthIntersections/fhirsrvr>.
- HL7 (2023). HL7 FHIR Standard. <https://hl7.org/fhir/>.
- HL7 (2024a). FHIR Tools Registry. <https://confluence.hl7.org/display/FHIR/FHIR+Tools+Registry>. Accessed: 2024-12-14.
- HL7 (2024b). Public FHIR Validation Services. <https://confluence.hl7.org/display/FHIR/Public+FHIR+Validation+Services>. Accessed: 2024-12-14.
- Hussain, M. A., Langer, S. G., and Kohli, M. (2018). Learning HL7 FHIR Using the HAPI FHIR Server and Its Use in Medical Imaging with the SIIM Dataset. *Journal of Digital Imaging*, 31(3).
- Kramer, E. (2024). Firely's FHIR .NET SDK. Open-Source Tooling, Webinar Series.
- Kramer, M. A. and Moesel, C. (2023). Interoperability with multiple Fast Healthcare Interoperability Resources (FHIR®) profiles and versions. *JAMIA Open*, 6(1).
- Lagger, Y. (2023). Vscode fhir tools. <https://github.com/laggery/vscode-fhir-tools>. Accessed: 2024-12-14.
- MITRE (2023). Ruby fhir testsript execution engine. <https://github.com/fhir-crucible/testscript-engine>.
- NIST (2024). NIST FHIR Toolkit. <https://github.com/usnistgov/asbestos/wiki/Introduction>. Accessed: 2024-09-05.
- Opie, C. A. (2024). Exploring security vulnerabilities in fhir server Implementations: a case study on ibm's fhir server in the context of the 21st century cures act.
- Otasek, D. (2024). The FHIR Validator. Open-Source Tooling, Webinar Series.
- Ozdemir, E. (2020). A General Overview of RESTful Web Services. In *Applications and Approaches to Object-Oriented Software Design: Emerging Research and Opportunities*. IGI Global Scientific Publishing.
- Scanlon, R. (2024). Inferno – FHIR Conformance Testing. Open-Source Tooling, Webinar Series.
- SOAPUI (2024). Getting Started with REST Testing in SoapUI | SoapUI. <https://www.soapui.org/docs/rest-testing/>. Accessed: 2024-09-05.
- Solbrig, H. R., Prud'hommeaux, E., Grieve, G., McKenzie, L., Mandel, J. C., Sharma, D. K., and Jiang, G. (2017). Modeling and validating HL7 FHIR profiles using semantic web Shape Expressions (ShEx). *Journal of Biomedical Informatics*, 67:90–100.
- Walonoski, J., Scanlon, R., Dowling, C., Hyland, M., Etema, R., and Posnack, S. (2018). Validation and Testing of Fast Healthcare Interoperability Resources Standards Compliance: Data Analysis. *JMIR Medical Informatics*, 6(4).
- Webber, J. (2010). *REST in Practice*. O'Reilly Media.