A continent to the state web service integration: a definition and the implementation approach

Boris Benko
boris.benko@kapion.eu
KAPION D.O.O.
Slovenia

Abstract

A new type of web service *IS* [1] integration is discussed, a continent to the state web service integration, dubbed as *k2s*. A definition of this web service type *IS* integration is given, and a general approach about the implementation is discussed. A brief comparison with other types of web service *IS* integrations is given, including but not limited to the global to the state web service integration, dubbed as *g2s*.

Keywords

continent to state information system integration, web services, digital inclusion, *k2s*

1 Introduction

In the global reach of *IT*, two general trends related to *IS* integration are emerging. On one side we have national and multinational companies, with a global reach, where *b2b* [2] integrations are taking place and also *b2c* [3]. On the other side we have a solid trend of public services to be offered in a digital form, to companies *s2b* [4] and consumers *s2c* [5], as well. As there is Single European Act [11] in force, it is clear, further integration processes are taking place.

An example of *k2s* system taken into consideration is *EU-CSW-CERTEX system* [8], commonly known as *CERTEX*. *CERTEX* is a system that connects customs systems with the EU's non-customs systems. This allows Customs authorities across EU to access relevant data within these non-customs systems. Quick access to non-customs systems is crucial for making informed decisions about whether or not to release goods for a specific customs procedure. While *CERTEX* system has several components, we'll be referring to the core *k2s* web service integration part in between non-customs systems on one side, e.g. the *k* part, against the EU member-state *IS*, e.g. the *s* part.

One example of a such non-customs system is *IS* for the importation of certain organic goods with a requirement of meeting the phytosanitary requirements, regulated as [9]. The *EU-CSW-CERTEX system* is in use as a mandatory requirement as of March 2025, but it is clear integration activities within *EU-CSW-CERTEX system* started much earlier, as the integration activities need to be implemented in each and every EU member state.

While the main topic is *k2s* systems, let's mention *g2s* integration system, as we will define it as well. Such an example is *UN/FLUX* standard, regulated as [4]. *UN/FLUX* facilitates the information exchange in between fishing domain entities. These fishing domain entities are fishing vessels, reporting the catch data to their domicile fishery authorities via different communication channels. The data received from fishing vessels at sea is then transferred to the geographically related authority via web services in a store and forward fashion. *UN/FLUX* system is comprised of many communication nodes, all communicating via *HTTPS/SOAP* messages.

2 Definitions

A continent to state web service integration system is the system, which integrates one supra-national authority, with multiple national authorities, to exchange relevant data via web services, whereby the data exchange is confined to a wider geographic region, possibly a continent. All stakeholders operate withing boundaries of their national jurisdictions.

A global to state web service integration system is the system, which integrates various entities, geographically dispersed, to one or possibly multiple state authorities via web services, whereby the data exchange is not limited by any geographic region. The stakeholders are normally bound by international standards and agreements and their jurisdictions are not playing the prerogative role.

3 Results

3.1 Specific requirements

The first and foremost requirement for *CERTEX* was that each software release has a status of *LTS*[6]. So, after the production tape-out it is expected for the version release to stay in production for years to come. There was a practical reason behind the requirement, as the complete EU member state *IS* is to be adapted to be compatible with *CERTEX*. Note, however, while we are discussing about *CERTEX* as *k2s* system, the *IS* of EU member state is really *s2b* or/and *s2c* to users in that particular member state. While the software maintenance for *CERTEX* was planned, retroactive functional upgrades for the version accepted in production were strongly discouraged.

The second requirement for *CERTEX* was two or more EU member states may communicate via *CERTEX*, even if they are at different software releases. It is possible, the web service integration may be degraded to be at the level of the lease capable software release, but in principle, EU member states should not be constrained in any way in intercommunication, while being on different *CERTEX* software releases.

The third requirement for *CERTEX* had a priority set to be more like a *nice to have*, but it was the integral part of the *CERTEX* success. Each and every EU member state may decide to upgrade

[1] The Information System
[2] A business to business integration.
[3] A business to consumer integration.
[4] A state to business integration.
[5] A state to citizens integration.

[6] A Long Term Support release.

Information Society 2025, Ljubljana, Slovenia

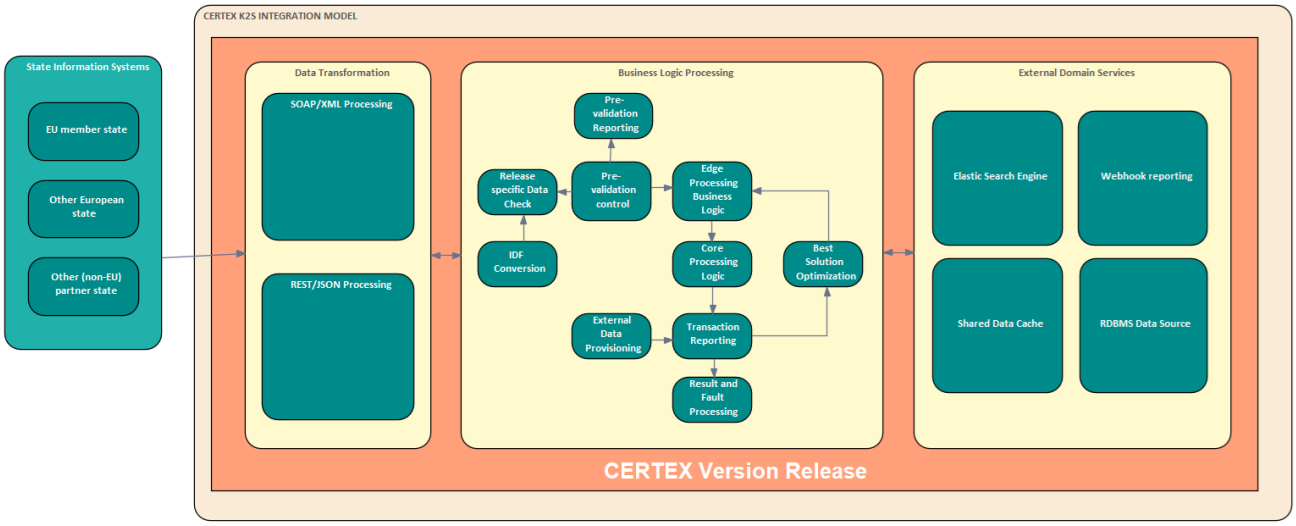

Figure 1: The *CERTEX* single release architecture

it's *CERTEX* release version either to the latest or to a higher version, at very least. But this web service release upgrade should be seamless, without the lengthy data migration, if possible.

3.2 An architectural overview

The complete *CERTEX* implementation relies on one software release, shown in Figure 1. Each release is done approximately every calendar year and since release version 1.0 in 2018, there are now six releases in production use, while the seventh release is in the development phase. All *CERTEX* releases run on *Jakarta EE* [3] server. While particular software releases are loosely coupled, the complete software package is of a monolithic type. From the year 2025 timepoint of view, the loosely coupled software releases would be very appropriate to be implemented with *Kubernetes* [5]. But as the software implementation started in 2018, at that time *Kubernetes* was not sufficiently mature and even if it would be mature enough in 2018, the decision of possible use of *Kubernetes* would require at least one prototype, which in turn, would require a delay of 12 months at very least. There was also an option to do a parallel software development on two different technology platforms - one *Jakarta EE*, as the proven one, and the challenger, namely *Kubernetes*. This, however, would increase the cost of the software development, at least in the initial phase, as two source codebases would be built in parallel. But even this parallel software development might uncover shortcomings on the challenger platform - *Kubernetes*.

3.3 XML Schemas

While the *CERTEX* has implemented the ability to communicate via *HTTP/SOAP* and *REST/JSON* messages, the bulk of messages is exchanged via *HTTP/SOAP*. There are two challenges present while messages are exchanged via *HTTP/SOAP* in between EU member states. One is, the standardized approach with *XML schemas* are needed, in order to standardize on a common messaging format with code lists included. There is a standard present *UN/CEFACT* for a global trade facilitation, defined as [10]. From the implementation perspective, *UNECE XML schemas* are well defined, but rather extensive in length and this presents a challenge to *Java* source code generators for *HTTP/SOAP* message processing. The challenge itself lies in the name-space tracking, whenever the *HTTP/SOAP* message request/response is generated. The *UN/CEFACT* uses name-spaces extensively and it is essential, name-spaces not in use, should not be present in the constructed *XML* message. Out of two *XML Java* code generators, namely *CXF* [2] and *Apache Axis2* [1], the former generates more optimal *XML* messages, while the latter does not track name-space usage very well and the corresponding *XML* messages are larger in size.

3.4 Record locking mechanism

The production version of *CERTEX* releases runs on a cluster of *Jakarta EE*, which are connected with a distributed shared cache. In order to facilitate a proper *ACID* [7] properties of transactional processing a capable record locking mechanism needs to be established. At first, *ACID* properties in *RDBMS* [8] were used. So, an *SQL* database was locking records, on as designed basis. This proved to be insufficient solution, as in the case when two or more parallel *HTTP/SOAP* web service calls were in progress, only the first commit of a web service call would pass, all other web service call transactions were rolled back. This proved to be insufficient from a business perspective. Namely every technical fault at transaction processing was reported to the EU member state authorities and a manual insight was dispatched to resolve the reported technical fault. These manual insights were costly from the human resources perspective, thus, a better collision resolution had to be found.

The solution with a distributed shared cache *Coherence* [7] was found. So, each *CERTEX* web service processing thread attempts to get a lock on the record. This action really implies creating a record in *Coherence* and obtaining a lock on the record in the shared distributed cache. If the *CERTEX* web service processing thread was not successful, a reasonably long lock 30 seconds timeout was used. The distributed shared cache locks proved to resolve processing collisions, however, in *k2s* web service integration systems a performant record locking mechanism is one of critical system components.

[7] Atomicity, Consistency, Isolation, and Durability.
[8] A Relational Database Management System

4 Discussion

In general, *k2s* web service integration systems are very large *IS*s, which connect the continent with states. And example of *CERTEX* integration is set forth and few challenges were present, while the software was developed. The underlying programming language and technologies need to be selected in order to implement a large scale *IS*, what *CERTEX* really is. *Jakarta EE* was selected for *CERTEX*, and this dictates the selection of other, compatible software components, such as *Coherence*. Selecting the *Jakarta EE* framework brings a lot of benefits into the project, such as a wide range of software vendors, even wider set of open-source alternatives. Furthermore, *Jakarta EE* is a widely used framework, thus a lot of practical software development answers can be found on Internet.

Kubernetes however, brings distinguished qualities, which *Jakarta EE* cannot match. *Kubernetes* brings efficient virtualization in a form of running containers within the pod. Furthermore, *Kubernetes* brings a computing resource dynamic scaling and declarative deployments. Furthermore, *Jakarta EE* is a server-side Java standard, comprising of a fixed set of Java specifications, such as *JAX-RS* [9]. This set of standards is versioned with the version of *Jakarta EE*. This is possible, with a distinguished Java library class-loading to upgrade a single standard. This is, however, an elaborate and unstable server configuration process, determining what are Java library class references and loading them, as well. Also, the *Jakarta EE* server includes all afore Java specifications - even if some or many Java specifications are not used, at all. Many software developers consider *Jakarta EE* as overly bloated and difficult to manage. And there isn't just the question of Java specifications, which are included into *Jakarta EE*, but also the inherent software security question. Related to the software security, we can follow a *less is more* imperative. So, less components the *Jakarta EE* server includes, less attack vectors are available to be attacked by hackers.

Is it possible to run *Jakarta EE* based software on *Kubernetes*? This is inherently possible, as many *Jakarta EE* servers maintain multiple managed servers, which may be run within the container. But this setup is not a true Kubernetes-native software program. A *Jakarta EE* compatible software is typically of a monolithic type, with a lot of possible Java modules, which are tightly coupled. A *Kubernetes* system typically inspires loosely coupled, micro-service based set of containers, with specific resource declarations.

It must be noted; *Kubernetes* is not on the same architectural level, as *Jakarta EE*. *Kubernetes* is in fact on the same level, as any enterprise-class, type-1 hypervisor, often referred as the bare-metal hypervisor. A true *Jakarta EE* counterpart is, for example, *Quarkus* [6] serverless environment. The problem with the *Quarkus*, as a viable alternative to *Jakarta EE* is, it is not the only serverless environment available and the list of Java standards available is less strictly defined.

The only natural path forward for *CERTEX* is to port the software from the monolithic *Jakarta EE* based form to the true Kubernetes-native, container-first software architecture. As *CERTEX* is a complex piece of software, the generalized methodology for porting monolithic Java server applications to Kubernetes-native, container-first serverless software. *KAPION R&D group* works extensively on the afore mentioned generalized methodology. A lot of development resources were invested into *Jakarta EE* compatible software. However, as *Kubernetes* offers clear advantages, compared to *Jakarta EE* compatible software, the generalized methodology as indicated above, is of a great interest to the Java software development community.

[9] Jakarta RESTful Web Services.

5 Conclusions

A new type of web service *IS* integration is present, a continent to the state web service integration, named as *k2s*. As the comparison reasons, a global to the state web service integration, dubbed as *g2s* was present as well. Definitions of both web service integrations were given and an example for *k2s* was present, as well. *CERTEX* is a system that connects customs systems with the EU's non-customs systems. Few *CERTEX* implementation challenges were discussed. For integration projects of such a scale, it is important the adequate *IT* software architecture is selected, where proven Java technologies took a precedence. Thus the *Jakarta EE* architecture was selected. It offers clear advantages in terms of using proven Java technologies, thus no delays are introduced into the software development timeline. However, in a sense disruptive *Kubernetes* emerged as a viable IT architecture substitute to *Jakarta EE* offering better scalability, and better declarative deployment capabilities, among other things. However, as both afore mentioned architectures, namely *Jakarta EE* and *Kubernetes* have a distinctive set of design patterns, a generalized methodology for porting the software from the monolithic server architecture to micro-services based, serverless software. *KAPION R&D group* is currently working on the generalized approach for the afore mentioned methodology.

References

[1] Apache. 2025. Apache axis2/java. Retrieved July 29, 2025 from https://axis.apache.org/axis2/java/core/.

[2] Apache. 2025. Apache cxf. Retrieved July 29, 2025 from https://github.com/apache/cxf/commit/main.

[3] Jakarta EE. 2021. Jakarta ee 9.1. Retrieved July 29, 2025 from https://jakarta.ee/release/9.1/.

[4] United Nations Economic Commission for Europe (UNECE). 2018. The un/flux standard. Retrieved July 29, 2025 from https://unece.org/trade/uncefact/unflux.

[5] Cloud Native Computing Foundation. 2014. Kubernetes. Retrieved July 29, 2025 from https://kubernetes.io/.

[6] Commonhaus Foundation. 2025. Quarkus. Retrieved July 29, 2025 from https://quarkus.io/.

[7] Oracle java. 2025. Oracle coherence. Retrieved July 29, 2025 from https://coherence.java.net/.

[8] European Parliament. 2024. Regulation establishing the european union single window environment for customs. Retrieved July 29, 2025 from https://eur-lex.europa.eu/eli/reg/2022/2399/oj/eng.

[9] European Parliament. 2019. The imsoc regulation. Retrieved July 29, 2025 from https://eur-lex.europa.eu/eli/reg_impl/2019/1715/oj/eng.

[10] UNECE. 2025. Unece xml schemas. Retrieved July 29, 2025 from https://unece.org/trade/uncefact/xml-schemas.

[11] European Union. 1987. The single european act. Retrieved July 29, 2025 from https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=legissum:xy0027.