Requirements of shared Data Management Services facilitating a Reference Architecture Realizing the Concepts of ECSS-E-TM-10-23

Workshop on Simulation for European Space Programmes (SESP)
24-26 March 2015

ESA-ESTEC, Noordwijk, The Netherlands

Tobias Hoppe[1,3], Harald Eisenmann[2]

[1]Airbus Defence and Space
FV Infrastructure, Engineering and Operations Products & Space Physics (TSOEC32)
Claude-Dornier-Straße, D-88039 Immenstaad, Germany
Email: Tobias.Hoppe@astrium.eads.net

[2] Airbus Defence and Space
Functional Verification & Engineering Infrastructure (TSOEC3)
Claude-Dornier-Straße, D-88039 Immenstaad, Germany
Email: Harald.Eisenmann@astrium.eads.net

[3]FZI Research Center for Information Technology
Intelligent Systems and Production Engineering (ISPE)
Haid-und Neu-Straße 10-14, D-76131 Karlsruhe, Germany
Email: hoppe@fzi.de

ABSTRACT

The European technical memorandum ECSS-E-TM-10-23 defines the concepts for a reference architecture for systems engineering covering a Conceptual Data Model (CDM). Engineering data suffers the diversity of tools from different vendors with the challenge to apply a CDM consistent across disciplines. ECSS-E-TM-10-23 is the foundation for information integration by specifying the semantics of collected and desired data instances. In this paper the necessary steps to realize a reference architecture and the relation to the concepts of ECSS-E-TM-10-23are presented. This is followed by a discussion about the impact of architectures on functionalities and vice versa. The emerging technology Open Services for Lifecycle Collaboration (OSLC) is introduced and its benefits and drawbacks are illuminated from an architectural point of view. OSLC is a technology to build a tool independent architecture supporting a flexible and customizable communication. In addition, the realized architecture can be build up depending on project-specific needs.

ISSUES WITH DISCIPLINE-SPANNING DATA EXCHANGE

Many domains are involved in current spacecraft projects dealing with a considerable large data exchange shaping a big data environment. These domains partially use specialized Commercial-Off-The-Shelf (COTS) tools coming from different vendors. That is why they have to handle different technologies, paradigms, file-formats, and semantics. The outputs of these tools as well as input documents are stored in domain-specific engineering databases. Consequently, the databases are designed with respect to the requirements of the COTS tools of a certain domain, but they reflect only a specific section of the space system development life-cycle.

Hence, there exists domain-specific Conceptual Data Models (CDMs) [12] making a semantic mapping of data between disciplines necessary. This is achieved by standard office products that are used to create documents being shared between domains. These documents are derived from the domain databases in manual processes. Fig. 1 illustrates data exchange based on standard office documents and commonly used text formats.

This results in inconsistencies between documents and databases making consistent data management covering the whole engineering process hard to achieve. In addition, data exchange is time-consuming and error-prone due to needed data transformations. Furthermore, the distributed data management leads to difficulties in getting a consistent domain-spanning dataset for integration tasks like simulations or system validations. Moreover, data is stored redundant in multiple domains. This provides a high potential for data inconsistencies.

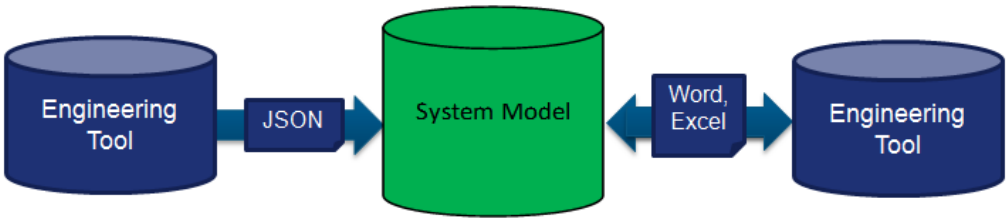

Fig. 1. Data interchange example realized by exchanging standard office documents and text documents.

Proceeding with model-driven engineering [13] requires a mapping of semantically concepts close to those of the application domain, because semantics are influenced by the used tools rather than the data itself. Moreover, information interchange with common semantic data normalization is needed [8], [9]. On top of this, multiple functionalities for discipline-spanning engineering tasks are needed, like data life-cycle tracking, change impact analysis, consistency checking as well as verification and validation.

Outline

In this paper different implementation strategies for reference architectures are discussed and their impacts on the needed functionalities are analyzed. In the following section the European technical memorandum ECSS-E-TM-10-23 [1] is briefly introduced focusing on the architectural implications of putting domain-specific data into a single data repository. Thereafter, the necessary steps to realize discipline-spanning reference architectures are explained. Afterwards, the impact on functionalities of different implementations of such architectures is discussed with special regard on overcoming the aforementioned issues concerning data exchange between COTS tools. In the final section, a conclusion is provided highlighting the missing elements of nowadays available frameworks for realizing the aforementioned architectures.

CENTRAL ASPECTS OF ECSS-E-TM-10-23 AND THEIR RAMIFICATIONS

This section briefly introduces the main concepts described by the technical memorandum ECSS-E-TM-10-23. Afterwards, the architectural implications of this standard are presented in detail.

ECSS-E-TM-10-23 is an emerging European standard facilitating consistent cross-discipline management of data. It addresses the aforementioned issues by providing an approach focusing on the alignment of the described patchwork of tools and databases being used in spacecraft projects [1].

COTS tools are build up on their own CDM and cannot be adjusted concerning data formats and tool interfaces. These CDMs are conflated into a single project-specific CDM which in turn is based on a global CDM specified in ECSS-E-TM-10-23. The global CDM reflects fundamental concepts of spacecraft system design. As a result, the specified global CDM is the foundation for a space system data repository that is used to store all information from all involved domains at a single place.

Architectural Implications using a global CDM as described by ECSS-E-TM-10-23

The usage of a global CDM as specified by ECSS-E-TM-10-23 results in multiple architectural implications. On the one side, a domain-spanning CDM and a related data repository implementing this CDM as defined by ECSS-E-TM-10-23 are the foundation for:

- A common knowledge of data and its structure by specifying the semantics of the individual elements.
- Prevention of redundancy through direct usage of data provided by other domains and storing results directly in the central data repository.
- Traceability of data changes, analysis of their impact, and consistent data management regarding both the consistency within a dataset from a single domain as well as domain-spanning consistency can be performed using the central repository's data from other domains.
- Getting a common baseline from multiple domains for domain-spanning tasks, like simulation, validation, and others without both manual gathering of data from multiple domains as well as without the pitfall of having inconsistent versions of data.

On the other side, even using a domain-spanning CDM as described by ECSS-E-TM-10-23 will result in discipline-specific CDMs due to discipline-specific adjustments needed to handle all aspects of a domain. Furthermore, the system development process has to be represented by project-specific adjustments in a domain-specific CDM. Data must be interchanged between domain-specific tools and a global system data repository. Therefore, the semantics of data from

domain-specific tools must be mapped to the global semantics and vice versa. In state-of-the-art frameworks this mapping is executed by some kind of adapter being part of importers and exporters. Fig. 2 illustrates this mapping based on an abstract example.

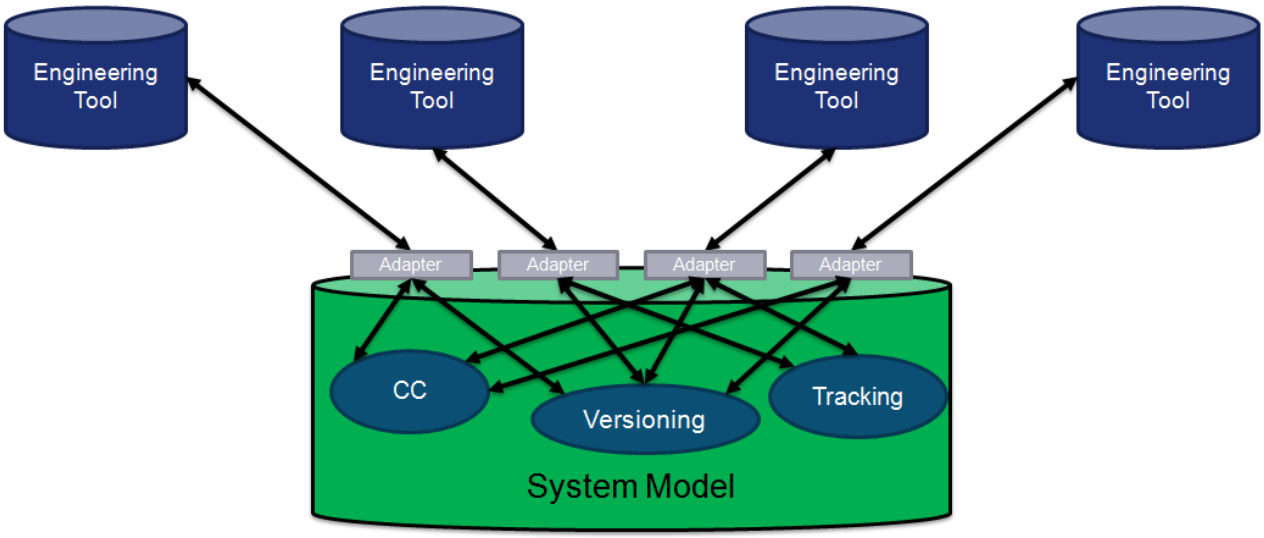

Fig. 2. The mapping of data between the engineering tool format and those of the central system model is realized in nowadays frameworks by adapters. They use internal functionalities of the systems engineering application to perform additional tasks, like consistency checks (CC), versioning, and data tracking.

REALIZING AN ARCHITECTURE IMPLEMENTING THE CONCEPTS OF ECSS-E-TM-10-23

In this section the general steps for realizing a framework implementing the concepts of ECSS-E-TM-10-23 are introduced. Additionally, the lost of semantics during working with data is inspected. This issue is also present even when a CDM is available and implemented by a framework.

Framework implementations that have to deal with a huge amount of data in a multi-discipline environment follow mainly the following steps during development [14]:

1. The data instances being collected and desired during runtime are described conceptually in a CDM. This supports getting a greater shared understanding of what data items mean and is the foundation for data integration as well as for a common data schema that will be used for data exchange. ECSS-E-TM-10-23 is an example for that kind of CDM.
2. Data integration using semantic data normalization must be implemented to integrate discipline-tools into the system database which is a foundation for discipline-spanning tasks like simulation runs.
3. The instance populations must be managed by several functionalities which allow working with the whole set of data. Among others, a global versioning system in form of common baselines is a key element for discipline-spanning functions. Further functionalities are for instance data migration and global data consistency checking.
4. Owing to semantics integration in a central repository further knowledge about the collected information can be gained by implementing additional functionalities like reasoning.

Causes for Lost Semantics during Working with Data even Using a global CDM

In this context it must be ensured, that the global semantics are equally interpreted by each domain. Otherwise, a central aspect of ECSS-E-TM-10-23 namely the preservation of semantics will be lost while working with data. It may result in data interpretation mismatches leading to system design errors. The central points to consider in this context are:

- Import and export of standard office documents can be performed by importers and exporters, but data from COTS-tools have different data formats, different paradigms, etc. Nevertheless, semantics of domain-specific data must be normalized to enable common data usage, even if proprietary formats needs to be analyzed or converted into a format that can be understood by a global system data repository. This offers a high potential for misinterpretation of data items being mapped to different concepts, especially if a corresponding concept will not exist on the other side.
- In relation to the last point, it must be ensured, that the semantics of all elements of a CDM are the same for each of the involved domains. Take for instance a system element position being exchanged between two

tools. If tool one takes a local coordination system and tool two expects positions in a global coordination system the positions needs to be transformed either during export from tool one or while performing the import into tool two. This depends on the semantics specified by the domain-spanning CDM, because it defines how a position is represented from a domain-spanning point of view.

- The project life-cycle is not respected concerning change propagation from one domain to another. This results in explicit actions concerning data actualizations instead of using notifications from resource management system to automatically notify observers about new versions of data.

A tool-specific metamodel and the part of the ECSS-E-TM-10-23 CDM being implemented by the corresponding tool differ in some points which leads to a semantic discrepancy between the applied metamodel and its counterpart in the CDM. Consequently, a semantic discrepancy between tools will exists whereby they are semantically harmonized in the CDM specified in ECSS-E-TM-10-23. This results in several issues concerning data exchange between tools as well as in relation with a systems engineering tool. As a result, nowadays frameworks implement individual adapters as shown in Fig. 2 to handle data exchange for each and every tool [6], [7].

IMPACT OF ARCHITECTURE ON THE IMPLEMENTATION

In this section different architectures implementing the concepts of the technical memorandum ECSS-E-TM-10-23 are presented and compared with each other. Thereby, the different aspects of realizing architectures with a central repository are highlighted. Afterwards, the emerging technology Open Services for Lifecycle Collaboration (OSLC) [3], [4][3] is introduced. Its contribution for data exchange, the implications on the architecture, the opportunities in context of semantically rich data exchange, and the missing parts for a fully-fledged infrastructure are highlighted.

From a very general point of view there exists only one feasible way to implement a reference architecture realizing the concepts of ECSS-E-TM-10-23, namely a central data repository with needed functionalities and the integration of engineering domains via well-specified interfaces. Architectures with a central repository are build-up following the hub and spoke principle with a central repository as hub and spokes as integration points for engineering domain tools. On the one side, this kind of architecture has multiple benefits [15]:

- A limited number of communication paths, namely exactly one for each integrated engineering domain tool.
- The dedicated communications paths lead to an efficient way of collecting needed information from a central repository and storing results in it.
- Spokes can be easily replaced which has no effects on further spokes.
- For information gathering it is not necessarily important to know the source of the information, because all information are available in the central repository and can be used directly.

On the other side, there are some drawbacks too:

- The central repository is a single point of failure which makes it necessary to implement further functions for data shadowing.
- In addition to the aforementioned point, the performance of a single central repository can be rather poor if data is not distributed on several machines accordingly.

Apart from this overall architecture view the needed functionalities can be implemented differently in a hub and spoke architecture to overcome the drawbacks of this kind of architecture depending on the individual needs. The two principally possible ways are:

1. A single repository implementing ECSS-E-TM-10-23 is used to capture the whole data from all tools of the involved engineering domains. Therefore, the repository needs to implement all needed functionalities and the engineering domains only provide the data as produced by their specific tools. The data exchange is realized by a common data schema. This centralized approach is illustrated in **Fehler! Verweisquelle konnte nicht gefunden werden.**. The advantages of this implementation are the presence of a central application performing all domain-spanning tasks and there is only one implementation for each function which reduces the chance of semantics discrepancies. The disadvantage of this approach is the strengthening of the drawbacks of a hub and spoke architecture as mentioned beforehand.
2. Compared to the first approach, the functionality could be mainly distributed on the engineering domains which are self-responsible for their specific tasks. The central repository only manages the data integration from the domains and is responsible for domain-spanning tasks. **Fehler! Verweisquelle konnte nicht gefunden werden.** illustrates this more de-centralized approach. The main advantage of this approach is the limitation of the drawbacks coming with hub and spoke architectures, because the performance is improved by delegating as much tasks as possible to the engineering domains. Moreover, the domains can partially work with their own data structures and are not forced to use the data structure coming from the central repository. This might result in less performance impacts. The most impacting disadvantage of using domain integration

with many functions is a hard replacement of domain integrations, because they need to implement all functionalities providing the same results for equal requests to the central repository. A further disadvantage of extracting functionality to domains is the high potential of semantic discrepancy between different implementations of the same functions in different domains.

It is not worth mentioning, that other architectures without a central repository will not provide a proper solution, because the involved tools in Model-Based Systems Engineering (MBSE) are far too specific [10]. In addition, domain-spanning functionalities like consistency checking are essential and in reality hard to be realized autonomously by engineering domains, because of a missing institution monitoring domain-spanning tasks. Moreover, those approaches would result in an infrastructure that is hard to adjust to project-specifics and that is hard to maintain regarding the change of used tools and changed communication paths. In contrast, the approaches with a central repository would enable domain-spanning data access which is performed automatically in the background by transforming domain-specific data into the format specified in the CDM and providing a domain-spanning view on the data.

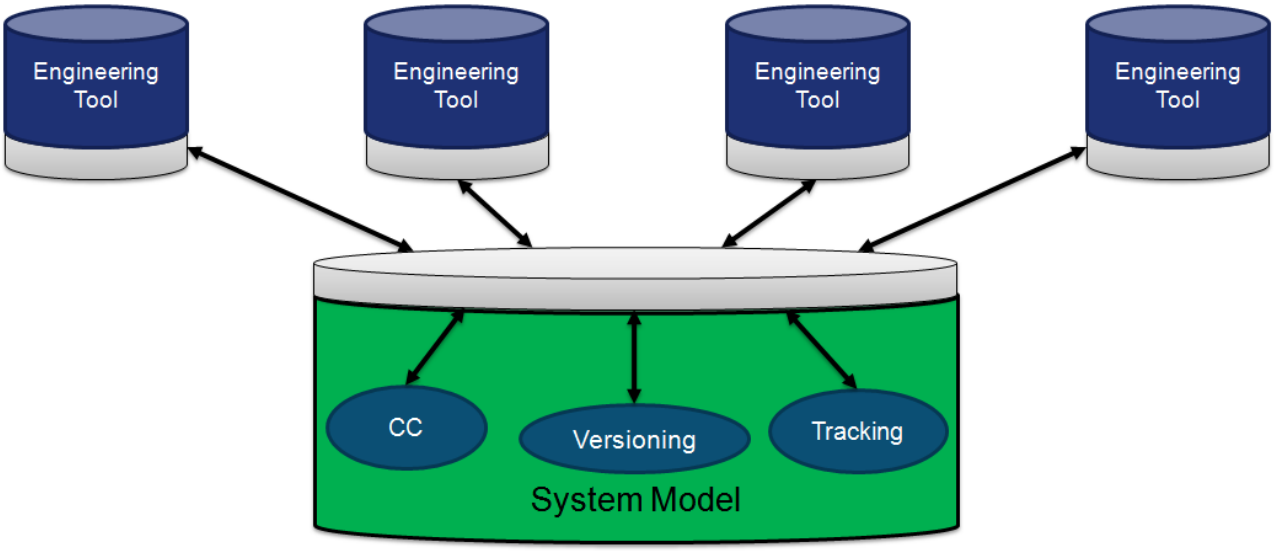

Fig. 3. An illustration of a centralized reference architecture for systems engineering. The systems engineering application represented by the system model implements all functionalities internally, like consistency checks (CC), versioning, and data tracking.

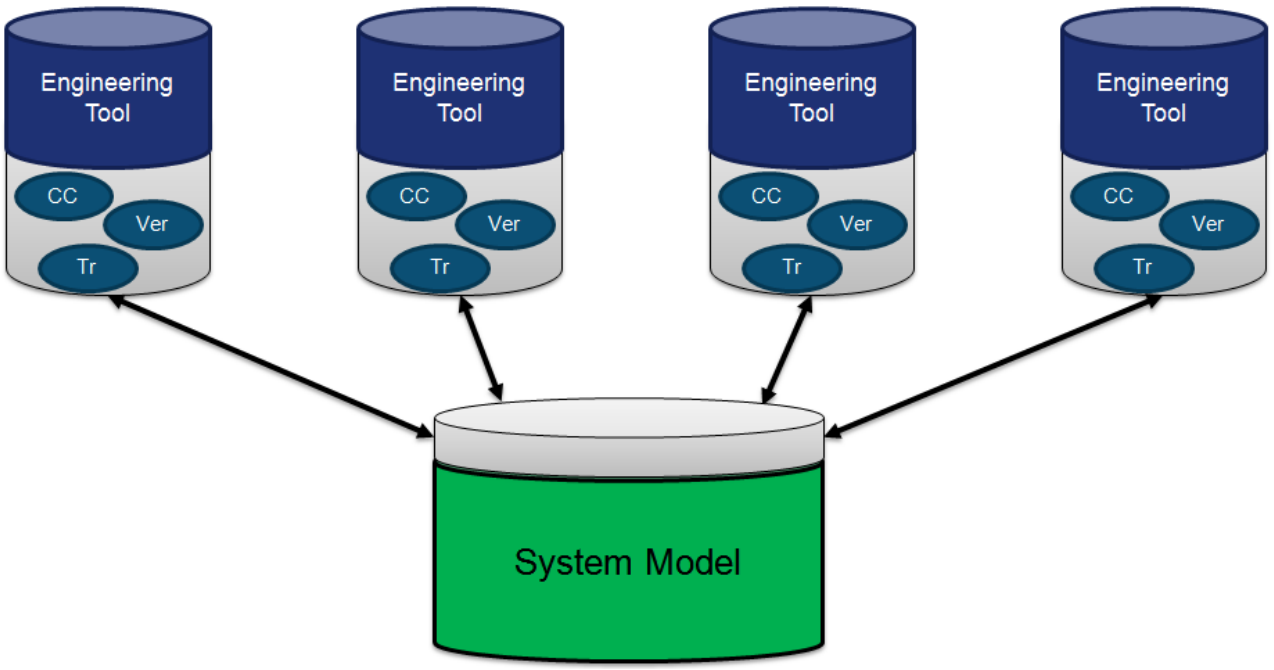

Fig. 4. An illustration of a de-centralized reference architecture for systems engineering. The engineering tool adapters encapsulate functionalities to perform additional tasks, like consistency checking (CC), versioning (Ver), and data tracking (Tr). Those functionalities must not be performed by the central application any longer.

Challenges to Meet during Architecture Implementation

Realizing a reference architecture implementing the concepts of ECSS-E-TM-10-23 requires the four steps described in the previous section, namely CDM development, data integration realization, data management, and data exploration functionality.

The first challenge is the implementation of the CDM as specified by ECSS-E-TM-10-23, because several of the modelled concepts cannot be taken over in the implementation due to the used technology for implementing the model as well as due to runtime performance reasons. On top of this, a mapping between the data as specified in a CDM and physical resources must be specified. This mapping is actually not present in ECSS-E-TM-10-23 and should be a part of it, because of the close relation to a corresponding CDM.

The second challenge to meet is the integration of engineering domain data into the central repository. Therefore, a resource exchange schema has to be derived from the data to resource mapping specified in the first step. This is the foundation for controlled data exchange as offered by emerging technologies like OSLC. A short introduction into OSLC will be given in the next subsection.

Depending on the chosen architecture the needed functionalities need to be implemented forming the central application with the central data repository (see central architecture **Fig. 3**) otherwise the central application is as much lightweight as possible and most of the functions are extracted into some kind of adapters for engineering tools forming a more de-centralized architecture as depicted in **Fig. 4**. Hereby, the benefits and drawbacks of de-centralizing a function shall be calculated carefully and further research is necessary to take the most proper decisions.

Finally, the last challenge is the realization of additional functionality being enabled by having all information in a central repository. Amongst others, reasoning might have a big impact on learning more about a developing system. Furthermore, property disjoint checks can be performed and ontologies might play a role.

An Introduction into OSLC

OSLC is an emerging technology to realize controlled big engineering information exchange. It is developed by an open community building practical specifications for integration of software components. The specifications build on the W3C Resource Description Framework (RDF) [11] whereby each resource is defined in terms of RDF classes and properties. A resource is each piece of information that can be exchanged, like text, xml, rdf, html, and many more text formats. Consequently, each resource is linked in some way to some internal block of information. The operations on resources are performed using the HTTP protocol and dependencies between resources are realized via links.

The OSLC framework realizes the principle of a service oriented architecture with a centralized registry where all services have to be registered and can be looked up afterwards. Further information about OSLC can be found on the community website [3].

OSLC as Foundation for Engineering Data Interchange

In the frame of engineering data exchange, OSLC provides a standardized protocol for information interchange between any kinds of engineering tools by tool-specific OSLC-Adapters. They encapsulate domain-specific tools from the environment and provide a common interface for data interchange [1].

OSLC is scalable regarding multi-user support and project-specific environments, because additional tools can be easily integrated into an already existing infrastructure by registering their OSLC-Adapters or by consuming data using additional OSLC-Clients. Additionally, OSLC-Clients can use multiple OSLC-Adapters to gather distributed data from multiple parts of the infrastructure. This is especially helpful to perform tasks relying on data from multiple domains, like simulation tasks. Furthermore, an OSLC-specification explicitly allows additional content being submitted in RDF-resources. This feature can be used to transmit customized elements. The OSLC-Adapter and OSLC-Clients transmit only RDF-resources being conform to the underlying specification.

Implications on the Architecture Realizing ECSS-E-TM-10-23 with OSLC

Realizing the concepts of ECSS-E-TM-10-23 with OSLC leads to a reference architecture for systems engineering supporting the following features:

- The OSLC-Adapters work as wrappers for the domain-specific COTS-tools and can transform tool-specific semantics into the global semantics of the space system data repository and vice versa. Therefore, a resource specification derived from the CDM specified in ECSS-E-TM-10-23 is needed to ensure semantic correctness of the data interchange between OSLC-Adapters and OSLC-Clients.

- Tool-specifics are encapsulated from the data exchange by a tool's OSLC-Adapter or OSLC-Client.
- The foundation for vendor-independent reference architectures will be available supporting a well-defined communication with any kind of tool being integrated via an OSLC-Client or OSLC-Adapter that implements the resource specification for ECSS-E-TM-10-23.
- The architecture is completely flexible regarding the communications paths, because each OSLC-Client gets delegated to the providing OSLC-Adapter by the OSLC-Adapter Registry.
- Additional tasks can be performed by OSLC-Adapters and OSLC-Clients, like consistency checks which might result in rejection of changes if a dataset became inconsistent by a change.
- The exchanged data can be tracked by an OSLC-Adapter to provide additional features for OSLC-Clients, like individual fine-grained notifications on data updates for those data being accessed by a certain OSLC-Client.
- Project-specific adjustments of RDF-Resources can be realized by optional parameters being explicitly allowed in the OSLC specification.

Nevertheless, OSLC does not provide any methodology to derive a resource specification from a CDM. Due to that, the first challenge of architecture implementation is still not meet completely. Tool integration can be performed based on OSLC whereby customized data structures cannot be specified semantically strong, because OSLC is based on fixed resource specifications being implemented by OSLC-Clients and OSLC-Adapters. Due to that, those resources cannot be changed during runtime and must be specified rather generic to handle all kinds of customization leading to a loss of semantics.

The OSLC-Clients and OSLC-Adapters can be easily extended by further functionalities working with both internal data structures of a domain as well as the semantically normalized data structures being exchanged. This allows centralized as well as de-centralized implementations. It only depends on how much of the functionality will be included in the central systems engineering application and which parts will be outsourced into the OSLC-Adapters and OSLC-clients of the other tools. The decision between a centralized and a de-centralized reference architecture must be taken explicitly for each functionality, because for some functionalities it makes sense to perform them using the locally available data, but for those functionalities based on widely distributed data a centralized realization on the systems engineering application will fit better. Even for some cases a mixture of both, will be an optimized solution.

CONCLUSION

This paper introduces different architectures for implementing a reference architecture realizing the concepts of ECSS-E-TM-10-23 and discusses their influences on the needed functionalities. Obviously, there is a need for integrating all tools of the domains being involved in the system development life-cycle based on MBSE. Therefore, different frameworks, even COTS-tools from multiple vendors supporting a variety of data formats, need to be integrated. The technical memorandum ECSS-E-TM-10-23 specifies a CDM for space systems engineering to form a common understanding of data over engineering domains and all involved institutions.

Realizing an architecture implementing the concepts of ECSS-E-TM-10-23 requires mainly four steps to be taken. The first one is mainly covered by ECSS-E-TM-10-23 by providing a CDM, but a derived resource description is missing. The second one focuses on tool integration. The last two steps deal with the implementation of needed and enabled functionalities. The implementation of each step has a significant impact on the resulting architecture making others than those using a central data repository impossible. The different manifestations of architectures with a central data repository have their own benefits and drawbacks. With regard on implementing the concepts of ECSS-E-TM-10-23 a more de-centralized approach putting as much functions as possible into the engineering domains keeps the responsibility for the data closer at the people having the most knowledge about it.

In this paper a closer look is taken into the emerging technology OSLC and its integration into as well as its impact on architectures. OSLC is a technology to build a COTS-tool independent reference architecture for systems engineering supporting a flexible and customizable communication with each registered OSLC-Adapter. Furthermore, the OSLC-Adapters can be enhanced by additional tasks, like data exchange tracking to inform data consumers about changes. Moreover, aspects, like version control and consistency checking, can be put into the OSLC-Adapters to pass over from common centralized architectures to more decentralized architectures to resolve the bottle-neck of a central application performing all tasks even not domain-spanning ones. In addition, the architecture can be built up depending on the individual needs of a project by adding or removing available OSLC-Clients and OSLC-Adapters from the current infrastructure.

Finally, OSLC is a step in the right direction, but still not enough. Several questions need to be answered before a reference architecture implementing the concepts of ECSS-E-TM-10-23 will be realized. The central points are the derivation of a data to resource mapping from a CDM, the handling of customizable data structures in correlation with semantically strong data exchange, and which functionalities can be implemented de-centralized.

REFERENCES

[1] T. Hoppe, and H. Eisenmann, "Realizing the Concepts of ECSS-E-TM-10-23 by Means of a Reference Architecture facilitated with OSLC", at *6th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, Stuttgart, Germany, October 08-10, 2014.

[2] ECSS European Cooperation for Space Standardization, "ECSS-E-TM-10-23A - Space Engineering - Space System Data Repository," *ECSS Secretariat ESA-ESTEC Requirements & Standards Division*, Noordwijk, The Netherlands, November, 25th, 2011.

[3] Open Services for Lifecycle Collaboration, Available at http://open-services.net/ [Online]

[4] OSLC Primer – Online book, Available at http://open-services.net/uploads/resources/OSLC_Primer_-_Learning_the_concepts_of_OSLC.pdf [Online]

[5] OSLC Core Specification Workgroup. OSLC core specification version 2.0. Open Services for Lifecycle Collaboration, *Tech. Rep.*, 2010.

[6] ESA, 2012, Virtual Spacecraft Design. Available at http://www.vsd-project.org/ [Online].

[7] H. Eisenmann, J. Fuchs, D. de Wilde, and V. Basso, "ESA Virtual Spacecraft Design", at *5th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, Lisbon, Portugal, October 17-19, 2012.

[8] H. P. de Koning, H. Eisenmann, and M. Bandecchi, "Evolving standardization supporting model based systems engineering", 2010.

[9] H. Eisenmann, J. Miro, and H. P. de Koning, "MBSE for European Space-Systems Development", *INCOSE Insight, 12(4):47–53*, 2009.

[10] S. Friedenthal, R. Griego, and M. Sampson, "INCOSE Model Based Systems Engineering (MBSE) Initiative", *INCOSE MBSE Track*, 2007.

[11] RDF Working Group W3C, RDF – semantic web standards, 2004, Available at http://www.w3.org/RDF/ [Online].

[12] J. Krogstie, A. L. Opdahl, and S. Brinkkemper, "Conceptual Modelling in Information Systems Engineering", *Springer*, 2007.

[13] A. G. Kleppe, J. B. Warmer, and W. Bast, "MDA explained, the model driven architecture: Practice and promise", *Addison-Wesley Professional*, 2003.

[14] A. Rosenthal, L. Seligman, and S. Renner, "From semantic integration to semantics management: case studies and a way forward", *ACM SIGMOD Record*, 2004.

[15] G. Hohpe, B. Woolf, "Enterprise integration patterns: Designing, building, and deploying messaging solutions", Addison-Wesley Professional, 2004.