

Taking Advantage of the Model: Application of the Quantity, Units, Dimension, and Values Standard in Concurrent Spacecraft Engineering

Volker Schaus, Philipp M. Fischer, Andreas Gerndt

German Aerospace Center (DLR), Simulation and Software Technology

Lilienthalplatz 7, 38108 Braunschweig, Germany

Email: {volker.schaus, philipp.fischer, andreas.gerndt}@dlr.de

Copyright © 2013 by Volker Schaus, Philipp M. Fischer and Andreas Gerndt. Published and used by INCOSE with permission.

Abstract. Designing a spacecraft involves many experts from different domains. In the early design phases they are brought together to discuss the spacecraft in a short period of time. The elements and building blocks the individual engineers want to use are stored and described in a shared system model provided by a software framework. A lot of features like the masses or dimensions are described using parameters. Due to the various domains and individual preference, many different units are used, leading to mistakes, distraction and misunderstandings caused by unit conversions. Our paper describes a pragmatic approach how the QUDV is integrated into an existing system model. This allows taking advantage of it by automatically analyzing the unit consistency as well as performing automatic conversions. The presented work shows that with a profound integration into the existing software framework, the engineers can overcome the error-prone task of unit checks and conversion problems.

Introduction

Dealing with scientific units and quantities is part of the everyday work of design engineers. Unit checks and conversions are routine tasks; engineers gain confidence in their results by comparing the calculated value to previous ones and their experience. If for example the order of magnitude of a result is the same as a previously calculated value of a similar problem, the engineer might consider this value as plausible. This can be described as *feel at home* in a system of units. Contrary, a misunderstanding on the units or a false conversion can implicate major design changes. Even worse, if it remains unnoticed, it can lead to mission failures, such as the loss of the Mars Climate Orbiter (NASA 1999). As a central example, one conclusion of the investigation report on that particular mission was that errors can happen, but Systems Engineering should provide adequate means to identify unit conversion problems throughout the whole lifecycle.

Looking at the lifecycle of such space systems most of the designs starts in so called concurrent engineering facilities (CEF) (Braukhane, et al. 2012). These facilities enable a design team consisting of various individual experts to catch up in a common place and to discuss the system design. Based on an agile approach and joint sessions that increase the common understanding for the design on system level are alternating with individual work where an expert can define a subsystem in more detail. Software tools like the Virtual Satellite (Deshmukh, et al. 2013) from the German Aerospace Center (DLR) or the Integrated Design

Model (Bandecchi, et al. 2000) from the European Space Agency (ESA) provide the backbone to these studies. Within these tools the engineers can store and exchange important facts in a shared system model. The engineering process itself is characterized by fast iterations and high level of communication across the design team. Many questions are raised across the technical domains and need quick answers in order to have progress in the study. The implemented system model tools supports this by sharing and reusing data across the whole design team, but by the natures of the various expert domains, one and the same data may be used and interpreted using different units. Since time is precious in such studies, the design team has to focus on solving the actual design problems rather than fiddling around with units and dealing with conversions. Therefore, already at the very beginning the demand for smart tools that take care of units and avoid confusion is inevitable.

A key element to encounter the unit dilemma is the conceptual model of *Quantities, Units, Dimensions, and Values*, defined in the appendix D5 of the SysML™ Standard (Object Management Group 2012). This model offers all the semantics and relations of units starting with the seven SI base units (Bureau International des Poids et Mesures 2006) and the quantity standard ISO/IEC 80000 (International Organization for Standardization 2009). On top of those basic definitions more complex relations are possible like the definition of a derived unit torque in Newton meters or prefixed units like nanometers. Together with a provided algorithm this model allows for automatic unit conversions and sanity checks as demanded by the concurrent engineering tools.

To bring the advantages of QUDV into the early spacecraft design, the system model of Virtual Satellite has been extended with this data model. It is based on Java/Eclipse and the Eclipse Modeling Framework (EMF) (Fischer, et al. 2011) which leads to the demand of an implementation of the QUDV in a matching format. In order to take full advantage of the QUDV and its connection to the existing system model, the algorithms to check and convert units automatically in the background are implemented as well.

This paper pinpoints the advantages of using QUDV in the early design phases and the concurrent engineering in particular. Thereafter, it discusses the model implementation using EMF as well as its integration with the already existing system model. The algorithms evaluating the unit model are explained in detail. They are the essential part needed on top to let the engineers benefit from the unit modeling. The paper concludes with describing the practical uses case of applying QUDV to early spacecraft design.

EMF Implementation of QUDV and System Model Integration

In order to make the QUDV accessible in the early design phases Virtual Satellite's data model needs to be extended in several ways. First of all, the QUDV model needs to be implemented using EMF. Second of all, it needs to be connected to the system model. In the presented implementation, the QUDV and the system model are individual models, but the system model is maintaining links to the QUDV. This means in the world of Eclipse that it is implemented as an independent set of plug-ins. It is a modular approach allowing reuse of the QUDV in various projects.

Virtual Satellite's System Model. The system model of Virtual Satellite is designed to support the engineers work. One instance of it is running on each workstation within the CEF. Modifications of the model are exchanged using a version control system. Figure 1 provides a simplified class diagram introducing the main elements and dependencies. All the data of the study is stored in a so called *Repository*. The model itself provides functionalities to

decompose the system as well as to save individual data attached to each of such *SystemComponents*. The decomposition is achieved by a parent child relation that allows each *SystemComponent* to contain further multiple *SystemComponents*. This allows the engineers to subdivide the investigated spacecraft into so called elements like the power or data-handling subsystem. Usually this is prepared by the team leader of the study, but the individual domain experts use the decomposition to individually describe necessary equipment and sub-equipment such as computers, sensors or solar arrays. The design information is stored as *Parameters*. Each *SystemComponent* may contain an individual set of *Parameters*. The characteristics of the *Parameters* are depending on the study itself and are usually agreed on beforehand. The engineers can assign a value to each parameter to represent properties like the mass or the dimensions of equipment or the total power demands of their respective sub-system. To achieve the accumulation of parameters and their values on sub-system or system level the model provides *Calculations*. The user defines *source* and *target* relations which are pointing to parameters. Additionally the type of *Calculation* can be specified. A solver performs the evaluation of the calculations, for instance calculating the sum of several parameters. When finished, the results are written back to the system model (Fischer, et al. 2011) (Schaus, et al. 2012).

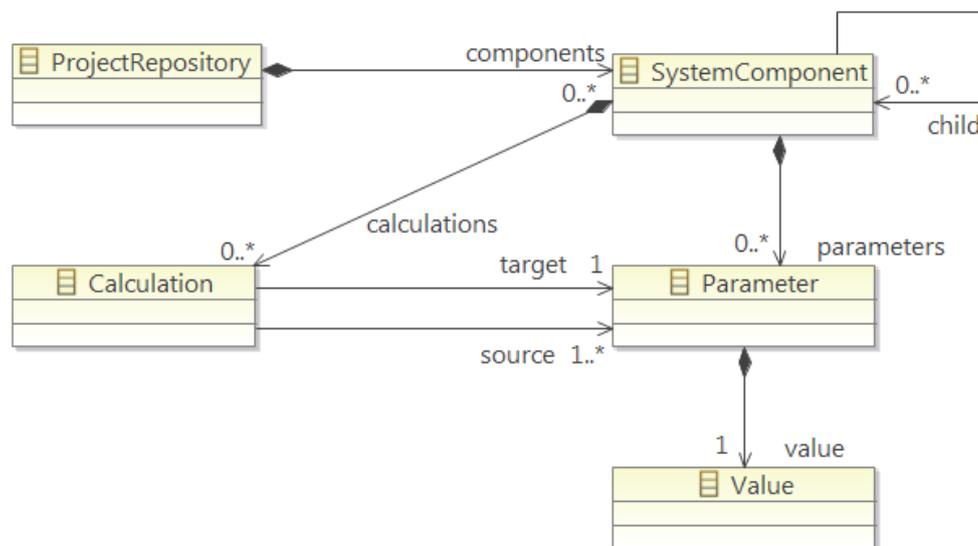


Figure 1. Class diagram of the internal system model of Virtual Satellite.

EMF Implementation of the QUDV. The EMF implementation of the QUDV was performed by redesigning the class diagram following the descriptions of the standard. EMF uses Ecore class diagrams to represent the model and it provides code generators to easily create a Java implementation of the data model (Steinberg, et al. 2009). Based on the EMOF, the Ecore models offer all the features necessary to implement the standardized QUDV model. As mentioned earlier, all the implementation has been targeted into an individual set of plug-ins focusing on the QUDV model and its functionalities only. Keeping it separated from Virtual Satellite’s system model allows for a later reuse in several different Eclipse based projects. The set of plug-ins follows the Model-View-Controller pattern. One plug-in contains the EMF Ecore model and the generated code itself. Another one is the controller which is handling the interactions with the data model. Yet another plug-in defines the user interfaces that provide a set of wizards offering a simple way to add and/or remove units from the QUDV model. Last not least, a helper plug-in holds the algorithms to perform the unit handling and conversions.

QUDV Integration with the System Model. The integration of the QUDV model and the corresponding plug-ins is constituted on various levels. Framework-wise, the newly created QUDV plug-ins have been added as integral part of the Virtual Satellite. This is naturally supported by the underlying OSGi/Equinox architecture enabling a plug-in reuse concept (OSGi Alliance 2012) (Clayberg and Rubel 2006). The integration allows other plug-ins like the one containing the system model to access and use the new QUDV features. Figure 2 depicts the actual integration on model level. The left side illustrates the system model with the aforementioned *ProjectRepository*. It has been extended to contain the *SystemOfUnits* from the QUDV within its *UnitManagement*. The *SystemOfUnits* is illustrated on the right hand side of Figure 2 and shows the many necessary classes defining the QUDV in EMF Ecore. The actual containment relation between the two models allows the storage of specific sets of units depending on the studies. Within Virtual Satellite the dedicated team leader is in charge of maintaining and updating the set of units whereas the software takes care of distributing the *UnitManagement* in the shared CEF environment. Additionally, each new study is initialized with a standard set of units. In order to allow the individual engineers to make use of the units, the *Value* of the *Parameters* have been extended to reference the abstract super class of all units *AUnit*. All other units inherit from this super class. In practice this means that each engineer can specify a unit together with each *Parameter* referencing to the accompanied set of units.

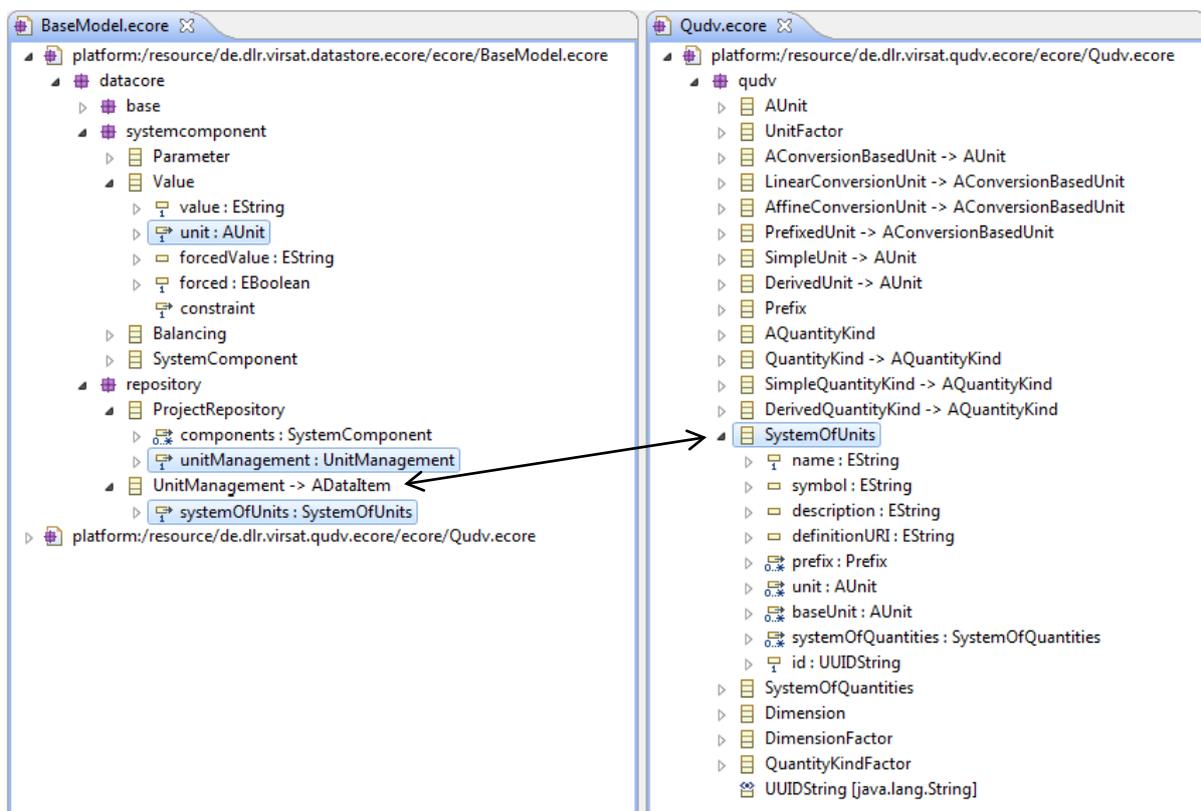


Figure 2. The Ecore implementations of the system model and the QUDV model

Algorithms for Unit Checking and Conversion

The aforementioned implementation of the QUDV model as well as modifications to the system model allows the engineers assigning units to their design parameters. But so far this just concerns the modeling side and is not exploiting the full potential such as automatic unit checking. To achieve the next step of taking advantage of the modeled units an algorithm that knows how to read and understand the semantics is mandatory. In conjunction with the

Calculation types of the system model it needs to analyze *source* and *target* *Parameters* and make sure that they fit correctly together. It implies that a multiplication of a force in Newton times a lever with a certain length leads to a torque in the appropriate unit. This method is implemented in three steps. The first step processes all *source* *Parameters* one after the other. Each *Parameter* is checked for its source unit and is transferred to its SI components which are stored as *SimpleUnits*. Necessary scaling and conversions on the *Parameter's Value* are applied as well. The second step merges all source *Parameters* depending on the type of calculation as well as it is taking care of reducing fractions. The third step finalizes the conversion by transferring the merged SI-units and converted value back to its target unit.

Step 1 – Factorizing Source Parameters to SI-Units. Figure 3 and Figure 4 outline the algorithm for the *Parameter's Value* conversion and Unit factorization. Figure 3 shows how the *Value* and its according unit is given to the algorithm. Depending on the type of unit the algorithm performs the conversion in five different ways. The first category of these five options handles the *SimpleUnit* itself. In case the source *Parameter* assigns already a *SimpleUnit* no further conversions need to be applied. The second category describes the conversion from an *AffineConversionUnit*, a *LinearConversionUnit* or a *PrefixedUnit*. In all three cases the offsets and factors are applied to the *Value* and the algorithm gets recursively called with the type of *AUnit* the current unit is derived from. The third category describes *DerivedUnit* which allows creating a kind of composition of all other units. Figure 4 shows this nested algorithm in detail. It is first parsing all referenced units and then calling itself recursively on each of them one after the other. It also takes care of applying potential exponents as well accumulating the right conversion that need to be applied to the *Value*.

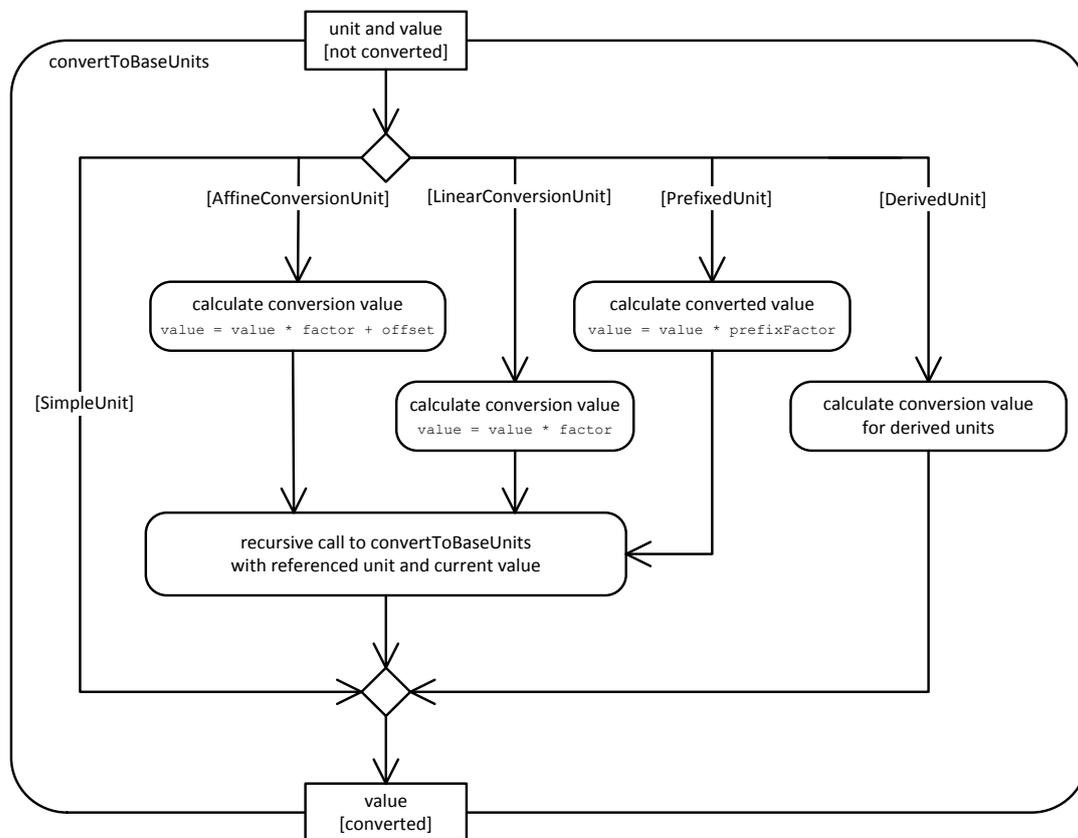


Figure 3. UML activity diagram showing the algorithm to convert a given value with associated unit recursively to a value with just base units.

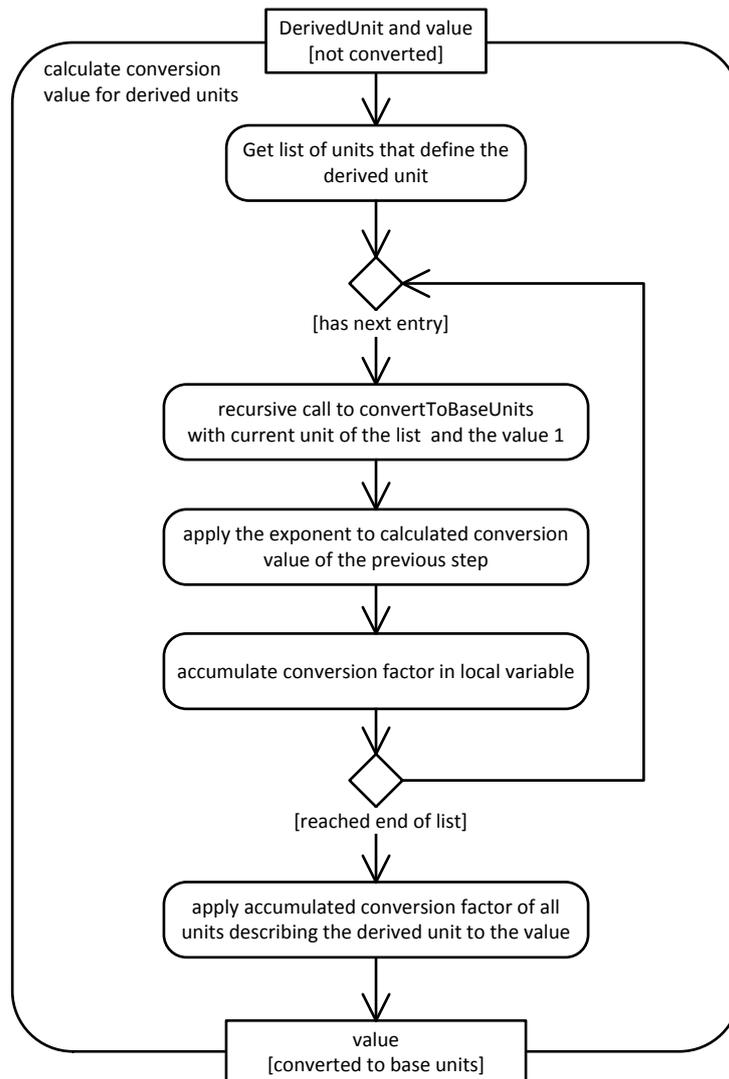


Figure 4. UML activity diagram showing the algorithm to convert a given value with associated unit recursively to a value with just base units.

Step 2 – Merging SI-Units of Source Parameters. In case the conversion is applied to either an addition or a subtraction merging the units is not necessary. It is enough to check the source and target units' *QuantityKinds* for equality. For example, a calculation adding up the masses of a spacecraft should verify that all inputs are mass quantities. The conversion that has been applied before already assures that all *Values* and *Units* are converted to *SimpleUnits*. In case of a multiplication or a division, a more complex algorithm is needed beforehand. Considering two multiplicands are used and have to be combined, their *QuantityKinds* have to be added up. Accordingly, when doing the division, they have to be subtracted. It means that the *QuantityKinds* of the *SimpleUnits* are merged with respect to the specified exponents. This is important to show that an area A which is the multiplication of two lengths L , sums up to the volume V which is equal to a cubic length L^3 . Once all *QuantityKinds* and their exponents are merged the algorithm has to check for *QuantityKinds* with a combined exponent of 0. Similar to the dimensionless *QuantityKind* they are not important for the subsequent conversion and have to be removed. Besides the checking and merging of the units and their respective *QuantityKinds*, the algorithm also has to add, subtract, multiply or divide the associated *Values*.

Step 3 – Factorizing Merged SI-Units to Target Parameter. Right after the units, their *Values* and the associated *QuantityKinds* have been calculated, factorized and merged, the result of the calculation has to be transformed back to the unit of the *target Parameter*. In fact this is performed analog to the algorithm described in Step 1, but in inverse order. The inversed algorithm is fed by the *target* unit as well as the result *Value* of the actual *Calculation*. The inverse algorithm factorizes the unit to its *SimpleUnits* using again the recursive approach. However, instead of applying the unit’s factors and offsets before the recursion they are applied afterwards. This is necessary to respect mathematic laws like operator precedence since obviously 10kg times 3 plus 2 kg is not the same as 10kg plus 2kg times 3.

Usage of QUDV in Virtual Satellite

Together with the implementation of the model as well as the algorithm to check and convert the units, it is also important to provide the appropriate user interfaces. Figure 5 shows the application Virtual Satellite with an opened study. The navigator on the left hand side reflects the *ProjectRepository* from the system model and the decomposed spacecraft. Additionally, it shows an item Unit Management which is opened in the editor on the right hand side. The editor provides access to the units as well as to the quantities.

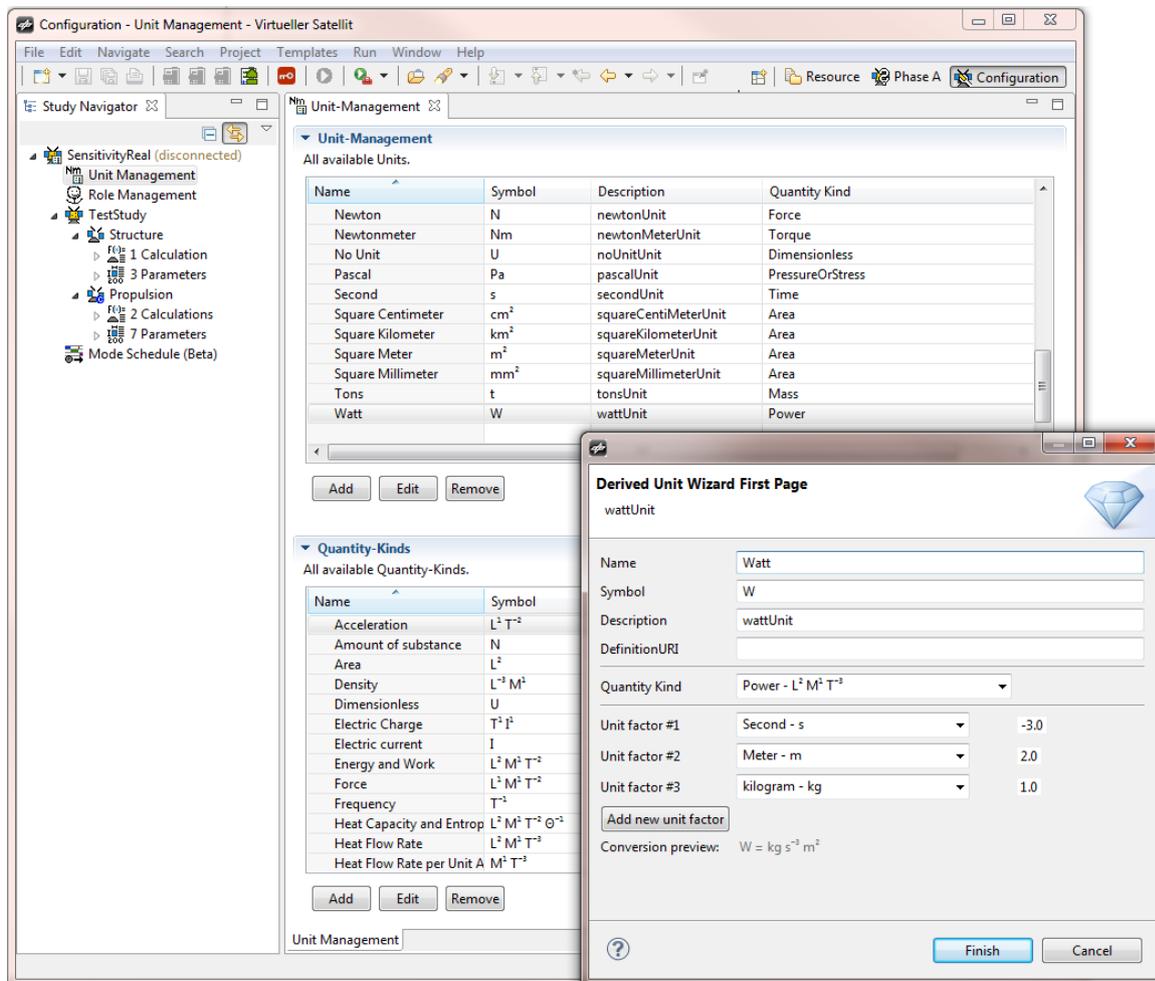
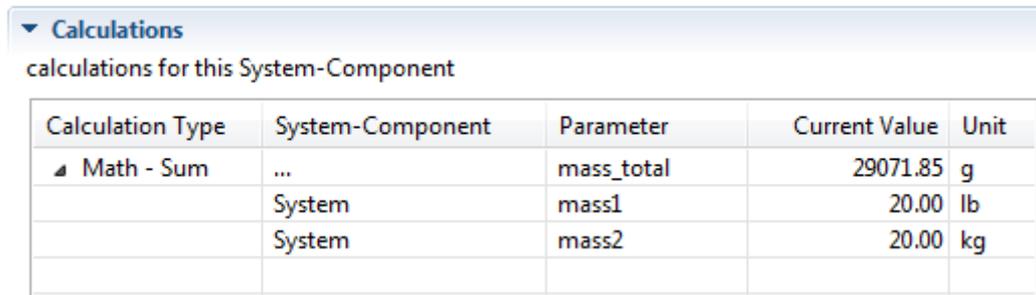


Figure 5. Screenshot of the software showing the management of Units and Quantities and the Wizard to create and change them.

Only the team leader has the rights to apply changes to the Unit Management by adding removing or altering units. As mentioned, the QUDV implementation provides wizards that help the team leader to enter new units. The wizard in the foreground shows an example of adding a *DerivedUnit*. Besides entering the units and also applying those to the parameters the engineers need to be provided with appropriate messages concerning issues with the units. Two use cases will illustrate how these mechanisms are implemented into the software.

Use Case I – Unit Conversion. Figure 6 illustrates the example of unit conversion. This is a typical issue that arises in international projects. Different nations use different units for various reasons; just agreeing on using the same unit like kilograms might confuse engineers which are trained in using pounds. This is also apparent for datasheets which often reflect national customs in the used units. Therefore the application of QUDV allows each engineer to work in the unit they are confident with, thus avoiding unnecessary conversion mistakes. The example shows an addition of two masses in two different units summing up to an overall mass converting to yet another unit. Since units can be specified individually for each parameter, the engineers are free to decide which unit they want to use. Additionally, they are able to change the output unit of a calculation according to their preference.



Calculations				
calculations for this System-Component				
Calculation Type	System-Component	Parameter	Current Value	Unit
Math - Sum	...	mass_total	29071.85	g
	System	mass1	20.00	lb
	System	mass2	20.00	kg

Figure 6. The Calculations table shows one entry that sums kilograms and pounds. The result is displayed in grams.

Use Case II – Unit Checking. Figure 7 shows the use case of applying an incorrect unit within the calculation. Same as the other example it is an often appearing case where incorrect values are applied to calculation by accident. The example shows that an input parameter's unit is changed to a force instead of a mass. The check of the *QuantityKinds* as described in the second step of the algorithm is realizing this issue and is reporting a problem through the graphical user interface. Since the problem view is part of each Virtual Satellite instance within the CEF an issue in the system model is addressed to all engineers in the session, thus reduces the risk of being disregarded.

Calculations				
calculations for this System-Component				
Calculation Type	System-Component	Parameter	Current Value	Unit
Math - Sum	...	mass_total	29071.85	g
	System	mass1	20.00	lb
	System	mass2	20.00	N

Parameters	References	Meta Data	Excel	Calculation
------------	------------	-----------	-------	-------------

Problems	
0 errors, 1 warning, 0 others	
Description	Location
Warnings (1 item)	
Found Unit/Quantity Problem: Math - Sum	System Balancing: mass_total = Math - S

Figure 7. The same addition as before, but mass2 is given in Newton now. The automatic unit/quantity check raises a warning in the problem, pointing the user to the right location to resolve this issue.

Conclusion

As introduced, just having the QUDV model applied to a system model is not sufficient. Modeling itself offers the baseline for consistency, but it needs some checking as well. Accordingly, the QUDV needs to be well integrated into the application where it is used. This includes capabilities for the modeling itself but also algorithms for the verification of what has been modeled using the units. This paper highlights the approach of integrating the QUDV into the software Virtual Satellite. Being based on Eclipse and EMF, the QUDV is also implemented as Ecore model. EMF allows connecting and integrating the QUDV easily into the existing system model. Additionally, wizards have been implemented as well to give an easy to use and accessible interface to the engineers to either add new units or to actually assign them to parameters in their design. Beyond the point of pure modeling, we showed how it is possible to take advantage of the modeled unit relations by automatic unit checking and unit conversions. The algorithm behind that functionality is based on a pragmatic approach staying completely in the background of software. Only in case the algorithm detects issues while checking the coherence of input and output parameters of a calculation it informs the design team through the graphical user interface. Altogether this work shows that the integration of the QUDV standard is a true advantage for the system modeling and realizable following a pragmatic approach.

References

- Bandecchi, M., B. Melton, B. Gardini, and F. Ongaro. "The ESA/ESTEC concurrent design facility." *Proceedings of the 2nd European Systems Engineering Conference (EuSEC)*. Munich, 2000. 329-336.
- Braukhane, Andy, Volker Maiwald, Dominik Quantius, and Oliver Romberg. "Statistics and Evaluation of 30+ Concurrent Engineering Studies at DLR." *Proceedings of the 5th International Workshop on System & Concurrent Engineering for Space Applications (SECESA 2012)*. Lisbon, 2012.

Bureau International des Poids et Mesures. *The International System of Units (SI)*. 8th edition. 2006.

Clayberg, Eric, and Dan Rubel. *Eclipse: Building Commercial-quality Plug-ins*. 2nd edition. Addison-Wesley, 2006.

Deshmukh, Meenakshi, Volker Schaus, Philipp M. Fischer, Dominik Quantius, Volker Maiwald, and Andreas Gerndt. "Decision Support Tool for Concurrent Engineering in Space Mission Design." In *Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment*, by Josip Stjepandić, Georg Rock and Cees Bil, 497-508. London: Springer, 2013.

Fischer, Philipp M., Volker Schaus, Daniel Lüdtkke, and Andreas Gerndt. "Design Model Data Exchange Between Concurrent Engineering Facilities by Means of Model Transformation." *Proceedings of the 13th NASA-ESA Workshop on Product Data Exchange (PDE 2011)*. Los Angeles/USA, 2011.

International Organization for Standardization. *ISO 80000-1 : Quantities and Units: Part 1: General*. ISO, 2009.

NASA. "Mars Climate Orbiter Mishap Investigation Board Phase I Report." 1999.

Object Management Group. *OMG Systems Modeling Language (OMG SysML™)*. Version 1.3. June 2012.

OSGi Alliance. *OSGi Alliance Homepage*. 11 7, 2012. <http://www.osgi.org/Main/HomePage>.

Schaus, Volker, Philipp M. Fischer, Dominik Quantius, and Andreas Gerndt. "Automated Sensitivity Analysis in Early Space Mission Design." *Proceedings of the 5th International Workshop on System & Concurrent Engineering for Space Applications (SECESA)*. Lisbon: ESA, 2012.

Steinberg, David, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF Eclipse Modeling Framework*. 2nd edition. Addison-Wesley Professional, 2009.

Biography



Volker Schaus is working as a research scientist at the DLR Institute for Simulation and Software Technology and is responsible for the Virtual Satellite project. He holds a Master Degree in aerospace engineering from the University of Stuttgart, Germany. He did his Master Thesis at the University of Sydney, Australia working adaptive neural network control algorithms for unmanned aerial vehicles. Before joining the DLR in 2010 he worked in VIP outfitting projects of wide body aircrafts as Mechanical Engineer. His research interests are digital product design and evaluation, Model-based Systems Engineering and Concurrent Engineering.



Philipp M. Fischer is currently employed as a research scientist at the German Aerospace Center (DLR) in the department "Software for Space Systems and Interactive Visualization" and is focusing on activities of model based software and systems engineering. He received his Diploma in electrical and computer engineering from the Leibniz Universität Hannover in Germany in 2007. Meanwhile he spent one year of studies in Computer Science at the Swinburne University of Technology in Melbourne, Australia. From 2007 until 2009, he supported Toyota's Formula One activities within the Department of Simulation and Performance Analysis at the motor sports headquarters in Cologne, Germany.



Andreas Gerndt is the head of the department "Software for Space Systems and Interactive Visualization" at the German Aerospace Center (DLR). He received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for different companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a post-doctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008.