Semantic Interoperability Centre Europe

Asset Development Guidelines

Issue: 1.3

Date: 2009-10-26

Authors:]init[AG Berlin

Document Change History

Date	Version	Author	Change Details
2008-09-25	0.1	Helmut Adametz, Andreas Billig, Sören Bittins, Jörg Caumanns, Jan Gottschick,	Initial Draft
2008-10-22	0.2	Helmut Adametz, Andreas Billig, Sören Bittins, Jörg Caumanns, Jan Gottschick,	Adding Migration and Use
01.03.2009	1.0	]init[AG Berlin	Upload to SEMIC.EU asset assistant
10.07.2010	1.1	]init[AG Berlin	Final preparation / document upload

The Asset Development Assistant is broken down into chapters. They present issues to take care of in the development of a semantic solution.

Table of Content

1 Requirements Analysis

It is essential to clearly define the requirements for your specific needs. Which data will be exchanged, and of what sort? It will help to avoid unnecessary effort as you will also recognise what is actually not required. Your interoperability asset(s) will serve a certain purpose. The more precise you can define this purpose, the easier it will be to meet those requirements.

1.1 Prepare a list of your requirements

1.1.1 Identify typical requirements

Consider these typical requirements and specify them should they be relevant to your data:

- Specific technical standards must be supported. Which ones?
- You are bound to certain legal frameworks and conditions. Which ones?
- The data shall be compatible to an existing data schema or certain data exchange standards. Which ones?

To get advice on the requirements in your case, please send a message to SEMIC.EU using the field below.

1.1.2 Consider non-functional requirements

Your proposed data structures may vary according to non-functional requirements like:

- data quality
- frequency and volume of data exchange
- security constraints
- given infrastructure

If applicable, specify these standards and parameters of the intended data exchange as additional requirements on your list.

1.1.3 List requirements from partners

If you are working in a cooperative project, take requirements from your partners into consideration and list them as well.
This should go as far as anticipating requirements from potential future partners to make your solution future proof. The granularity of data models is a critical factor here (see the section on semantic conflicts below).

1.1.4 List technological requirements

Collect standards, frameworks, and technical environments in your project's domain and environment at an early stage. They may be considered for additional requirements.

1.1.5 Consider additional requirements

Cross-border perspectives:
Even if, in the scope of your project, cross-border data exchange is not considered or very limited, it can still pay off to anticipate future scenarios of cross-border data exchange. Small investments in your project may substantially decrease costs in the future and ensure the future of your solution.

Multilingual perspectives:
Even if your project is monolingual in nature, you might consider to anticipate future pan-European use and thus prepare your solution for multi-lingual use. It is recommended to use a pivot language (e.g. English) that can be used to bridge the gaps between multiple languages. For further reference, see the SEMIC.EU Study on Multilingualism

1.2 Prepare a list of objects/attributes to be exchanged

What is the exact set of data that must be transferred?

1.2.1 Set up a list of objects/attributes to be exchanged

Be sure to limit the number of objects/attributes to a minimum. This will minimize possible semantic conflicts.
Use the question box for advice on the identification of required data.

2 Concept & Specification

2.1 Identify Required Artefacts

2.1.1 Decide on type of artefacts to fit all requirements

When designing the artefacts (e.g. UML-class, XML Schema file) pay attention to semantic conflicts: solve all conflicts as agreed and be sure not to raise new ones.
For information on risks and conditions, see the chapter on "Development of an asset".

2.1.2 Decide on optimum granularity of artefacts

An artefact should contain all information on the entity it models. It should not contain information that could be modelled as an entity which could exist independently.
E.g. the model of a person should not include the model of an address since the concept of an address is independent of a person.

2.2 Involve SEMIC.EU

In any case, using the visibility and starting points for collaboration via the platform is recommended. You should consider the following steps.

2.2.1 Register your project with SEMIC.EU

Registering your project with SEMIC.EU will attract the attention of similar projects with which you can realise synergies.

2.2.2 Register your asset with SEMIC.EU

Registering your asset with SEMIC.EU even at an early phase can also generate interest by similar projects and organisations. From your perspective, this is a means of getting ideas and early quality assurance from other users and the SEMIC.EU team.

2.2.3 Initiate communication with SEMIC.EU community

You may use the SEMIC.EU platform to initiate communication with national experts, domain experts, related projects, and the SEMIC.EU team. For matters that need discussion, the forum section should be used.

3 Reusing existing assets

It is always worthwhile to check if there is an existing solution that you can use as a starting point for your own solution. In fact, this is the basic idea of SEMIC.EU.

3.1 Is reuse possible?

Search, check against requirements, adapt – these three steps should be regarded in your reuse considerations.

3.1.1 Search for assets to be reused

Keywords, domains and free text search of the SEMIC.EU repository are ways to check whether there are assets developed by others which you can base your solution on.
If you are uncertain whether such an asset exists or if it can be transferred to your case, ask for advice by SEMIC.EU's repository managers.

3.1.2 Check reusable assets against requirements

- Define K.O. criteria for your decision.
  Example: You cannot understand the original language of the asset in question.
- List parameters for your decision: What does the asset have to deliver to be eligible as a model for your case?

3.1.3 Modify a reusable asset

- Delete unnecessary objects and attributes
- Add required objects and attributes
- Stick to semantic/syntactic rules (see below sections)
- Document any changes made

4 Development of a new asset

When you develop your asset from scratch you have several types to choose from.
The following list describes tasks and potential conflicts for one type of interoperability asset at a time.
If you know already which type of solution you are implementing, check the tasks of all other models. This will remove them from your list of "open issues".

4.1 Code Lists

4.1.1 Completeness of data types

All objects must be defined with data types whose implementation in XML Schema is clearly specified. If there are restrictions in length for certain values, they must become components of the data model.

Risks:

- If those who develop the models neglect information on data types because they consider it self-evident, for instance, that the title of a person must be of the type "string", this can lead to a conflict in case the title is implemented for a system as a code list. Post codes are another example as they vary from country to country and may consist of four digits or five digits, contain letters etc.
- Even if the data type is already specified in the names of the objects, (e.g. "StreetString", "BirthDate"), the explicit definition of the data type should not be omitted. This could highlight, for instance, that "StreetNumber" is not supposed to be implemented as a set of digits but as a string, instead.
- If in a data model the implementation of data sets in XML Schema is not determined unambiguously, the technical interoperability can be at risk despite (perceived) semantic interoperability. Example: In a data model, objects receive the data type "date". One implementation, however, does not use the XSD data type "date" in XML Schema but "datetime", in order to facilitate the representation of the time of day as well. Another implementation defines its own XSD data type based on string to be able to transfer incomplete dates. Seamless data exchange between these implementations is not possible.
- Missing indications of restrictions of length. Example: a long family name is transmitted but cut off at a maximum amount of characters before saving. If the truncated family name is transferred and the receiving system is unaware of the restriction of length, the matching data set cannot be identified.

4.1.2 Disjointedness of values

The potential values of a code list must be disjointed if only one value is allowed for the transfer.

Values which are not disjointed can generate different data sets despite identical information.

Example: The values "deleted" and "deleted by virtue of a warrant" are not disjointed, and a deletion based on a warrant can also generate the value "deleted". This arbitrariness leads to bad data quality because it is not certain whether the data sets "deleted by virtue of a warrant" are complete.

4.1.3 Versioning

Whenever dynamic code lists (i.e. code lists to which values can be added) are deleted or changed, they must be versioned. Any data set using the code list must also reference the version used. Any productive version of a code list ever released must be kept available in order to preserve the readability of old data sets.

Risk: A code list is changed and simply replaced.

Examples

- The value in an old data set cannot be interpreted any more.
- The value in an old data set has a new – incorrect – meaning.

4.1.4 Centralised maintenance of code lists

For the maintenance of a code list, a responsible authority must be assigned, irrespective of how many communication partners use the code list.

This is to avoid double attribution of keys in case of decentralised maintenance.

Another risk is that of overlapping extensions:

Partner 1, for example, adds "vessel" to a list of vehicles. Partner 2 extends the same list by adding the values "rowing boat", "sailing boat" and "power boat". This is also a problem in case of different keys since no unique association is possible.

4.1.5 Define a rollout scenario for new versions

There must be a pre-defined process among all communication partners for the transition to new versions of code lists.

It must be avoided that the sender uses the keys of a new version of a code list, which is not yet known to the receiver.

4.1.6 Handling of missing values

In case it is possible that new values are added to code lists which must be transmittable without delay (i.e. before the code list can be extended), the data model must provide for this scenario. As soon as the code list has been extended, existing data sets must be transformed to the new code. As a rule, this must be done manually, since the centralised attribution of new values will render an automated representation virtually impossible. For values previously not represented on the code list, an extra field must be dedicated.

Mind these risks:

- Missing information in data exchange if there is no field for missing values.
- Abuse of existing fields that have different semantic attributes in cases where no additional field for missing fields exists.
- Problems of interoperability if data sets which have been transferred with new values (instead of codes) are not adapted to the new code list.
- Use of more than one value for a single context in case the extension of the code list takes too long – lack of interoperability before the extension of the code list and the adaptation of all data sets.

4.2 UML models

4.2.1 Completeness of data types

All objects must be defined with data types whose implementation in XML Schema is clearly specified. If there are restrictions in length for certain values, they must become components of the data model.

Risks:

- If those who develop the models neglect information on data types because they consider it self-evident, for instance, that the title of a person must be of the type "string", this can lead to a conflict in case the title is implemented for a system as a code list. Post codes are another example as they vary from country to country and may consist of four digits or five digits, contain letters etc.
- Even if the data type is already specified in the names of the objects, (e.g. "StreetString", "BirthDate"), the explicit definition of the data type should not be omitted. This could highlight, for instance, that "StreetNumber" is not supposed to be implemented as a set of digits but as a string, instead.
- If in a data model the implementation of data sets in XML Schema is not determined unambiguously, the technical interoperability can be at risk despite (perceived) semantic

interoperability. Example: In a data model, objects receive the data type "date". One implementation, however, does not use the XSD data type "date" in XML Schema but "datetime", in order to facilitate the representation of the time of day as well. Another implementation defines its own XSD data type based on string to be able to transfer incomplete dates. Seamless data exchange between these implementations is not possible.

- Missing indications of restrictions of length. Example: a long family name is transmitted but cut off at a maximum amount of characters before saving. If the truncated family name is transferred and the receiving system is unaware of the restriction of length, the matching data set can not be identified.

4.2.2 Completeness of multiplicities

Relations between objects must be defined with all multiplicities.

In UML, it is impossible to leave the multiplicity undefined. Where nothing else is specified the multiplicity is interpreted as 1.

Omitting the multiplicity in the data model can cause severe problems in the eventual data exchange. Example: Requirements call for a multiplicity of 0..1 (the attribute may be missing) but the model does not spell out what exactly is interpreted as 1 (mandatory attribute). Accurate implementations of the model produce an error whenever data sets are delivered which lack the attribute.

4.2.3 Meaningful identifiers

The names attributed to UML elements must be as meaningful as possible to make the meaning or purpose of the object quickly discernable, e.g. "nameIndividualPerson". To avoid extensive identifiers, agreed abbreviations can be used.

Cryptic or overly simple identifiers (e.g. "name") for UML elements compromise the readability of the data model for humans.

4.2.4 Misspellings

Only thorough and targeted quality assurance can rule out (or minimise) wrong spellings in UML models.

Misspellings, such as scrambled letters, bear a hazard that developers "correct" these errors at implementation leading to incompatibility with unaltered implementations of the (erroneously named) element.

4.2.5 Use of Core Components

Search for suitable core components for objects of your own model.

New communication partners, who join after the model has been developed and implemented, bear the risk that bijective mapping in a common model is not possible. Example: Street number and street number affix are once modelled jointly and once separately. If both are based on the same core component, the mapping will work even if the attributes are named differently and there are additional attributes which are semantically disjointed from the attributes of the core component and which are only used by one partner.

4.3 XML Schema

4.3.1 Naming and Design Rules

Naming and Design Rules (NDR) facilitate technical interoperability between the different systems. But even for single systems, the implementation is made easier if all data models are subject to the same NDR.

- Semantic interoperability of data models alone does not safeguard technical interoperability. A single UML model can be mapped to different XML schemas. These schemas are not technically interoperable (for software).
- If systems that were developed simultaneously are to be connected for which no NDR were defined, the implementation of a mapping becomes more complicated and the performance of the mapping is slowed down.
- If no Naming and Design Rules are defined, very different approaches to XSD design can be chosen in the development and maintenance of XML schemas. That complicates the implementation and compromises the performance of the running system.

4.3.2 Uniform naming of data types

The readability of XML schemas can be enhanced by indicating data types in the names of XML elements.

Examples:

- a complex data type with several attributes for one address can be named "addressComplexType"
- a simple data type with only one value for post code can be named "postcodeSimpleType"
- a data type for a state the values of which are defined by a code list, can be called "countryCodeType"
- a data type for a status the values of which are defined by enumeration can be named "statusEnumerationType".

If the names of XML elements do not contain information about the respective data type, errors of software implementation in the programme code and in the created XML instances are hard to detect. Handling XSD becomes considerably more difficult for humans without the support of names indicating the data type.

4.3.3 Meaningful identifiers

The names attributed to XML elements, attributes and types must be as meaningful as possible to make the meaning or purpose of the object quickly discernable, e.g. "nameIndividualPerson". To avoid extensive identifiers, agreed abbreviations can be used.

Cryptic or overly simple identifiers (e.g. "name") for XML elements, attributes and types compromise the readability of data model for humans.

4.3.4 Misspellings

Only thorough and targeted quality assurance can rule out (or minimise) erroneous spellings in XML schemas.

Misspellings, such as scrambled letters, bear a hazard that developers "correct" these errors at implementation leading to incompatibility with unaltered implementations of the (erroneously named) XML schema.

The use of UML tools which generate XML schemas lowers the probability of typing errors because due to the object-orientation of UML tools a name is defined only once and is then reused everywhere by referencing. At the same time, a misspelling in a UML tool is copied to all the referenced points in a XML schema.

4.4 Core Components

4.4.1 Completeness of data types

All objects must be defined with data types whose implementation in XML Schema is clearly specified. If there are restrictions in length for certain values, they must become components of the data model.

Risks:

- If those who develop the models neglect information on data types because they consider it self-evident, for instance, that the title of a person must be of the type "string", this can lead to a conflict in case the title is implemented for a system as a code list. Post codes are another example as they vary from country to country and may consist of four digits or five digits, contain letters etc.

- Even if the data type is already specified in the names of the objects, (e.g. "StreetString", "BirthDate"), the explicit definition of the data type should not be omitted. This could highlight, for instance, that "StreetNumber" is not supposed to be implemented as a set of digits but as a string, instead.
- If in a data model the implementation of data sets in XML Schema is not determined unambiguously, the technical interoperability can be at risk despite (perceived) semantic interoperability. Example: In a data model, objects receive the data type "date". One implementation, however, does not use the XSD data type "date" in XML Schema but "datetime", in order to facilitate the representation of the time of day as well. Another implementation defines its own XSD data type based on string to be able to transfer incomplete dates. Seamless data exchange between these implementations is not possible.
- Missing indications of restrictions of length. Example: a long family name is transmitted but cut off at a maximum amount of characters before saving. If the truncated family name is transferred and the receiving system is unaware of the restriction of length, the matching data set cannot be identified.

4.4.2 Determining code lists

For all attributes of the type "code", you should determine or, at least, recommend concrete code lists

Risk:

- A lack of interoperability if the data model is equivalent but the used code lists are not. This means that mapping between values is not always possible.

Examples:

- Marital status: {married, divorced, widowed, unmarried} vs. {married, divorced before 1990, divorced after 1990, partner deceased, partner pronounced dead, unmarried}
- Gender: {male, female, unknown} vs. {male, male/formerly female, female, female/formerly male, undefined}

4.4.3 Completeness of multiplicities

Relations between objects must be defined with all multiplicities.

Omitting the multiplicity in the data model causes severe problems in the eventual data exchange. Example: Requirements call for a multiplicity of 0..1 (the attribute may be missing) but the model does not spell out what exactly is interpreted as 1 (mandatory attribute). Accurate implementations of the model produce an error whenever data sets are delivered which lack the attribute.

4.5 Mappings

4.5.1 Rounding differences caused by different data models

If calculations are required to add missing data, the formulas must be standardised. In addition, the data type of the calculated value of one model must match the data type of the not calculated value in the other.

Example:
One model saves net and gross price, another saves the sales tax rate and only the net price. This is semantically unambiguous and can be mapped reciprocally. In practice, however, there are difficulties since it is not defined onto which decimal place the gross price has to be rounded, esp. when sums are calculated. In this case, different implementations generate different gross prices despite semantic unambiguity.

5 Avoiding Semantic Conflicts

Semantic Conflicts are the most severe problems a project can face in the field of semantic interoperability.
Wherever natural language is used, semantic ambiguity is present. It must be contemplated in each case of data exchange whether these ambiguities are acceptable or whether they will eventually cause risks to the project.
In data processing, semantic ambiguities must be contained and ruled out wherever possible. Compared to humans, digital systems are far less capable of deciphering the intended meanings of signals (spoken or written language).

If your project requires data integration from different sources or an exchange of data between different systems, questions about the exact meaning, i.e. the semantics, of the data will arise inevitably.

5.1 Examples

Elements of two different data models use the **same label but with different meanings** (they refer to different things in the real world).

- The element "price" can use the same name for prices inclusive or exclusive of sales tax.
- An element "title" can contain values like "Mister", "Mistress" and "Professor", but in another model, the title can refer to a headline or the name of a publication ("Tales of Mystery and Imagination").

Differences in **granularity** of models. An element considered as a single item in one model is treated as more than one in another.

- There are two models containing the text element "family name". In one of the models this includes name affixes like titles of nobility, in the other, there is a separate field for these affixes.

Inaccurate naming of elements.

- In a model, the element "city" is of the type "code", and there is another element "city abroad" which is of the type "Text". The model is built on the assumption that the incorporated code list only covers domestic locations. It uses the field "city abroad" for any city which is not part of the code list. Yet, after a reorganisation of local government, there might also be domestic cities not covered by the code list. The field should, therefore, not be named "city abroad" but e.g. "city uncoded".

5.2 Impacts of semantic conflicts

Depending on the type of the semantic conflict (see below) and the data contained in conflicting objects and attributes, the impacts may range from mildly annoying to catastrophic. E.g. the NASA's "Mars Climate Orbiter" was lost due to a semantic conflict (different scaling and units for impulse data, metric vs. English units) resulting in the loss of more than US$ 300 million.

Before deciding on solution strategies for semantic conflicts it is therefore important to weigh the impact and consequences of the conflict. Depending on the type of the semantic conflict, different strategies are available. In general it comes down to defining a common semantic understanding and the ways to reach it.

Risks:

- If inventory data are migrated because of different levels of granularity of the communication partners, there is a risk that a future communication partner requires precisely the model that the data was migrated from.
- In cases of ambiguous semantics, a future data exchange with partners who have an unambiguous model becomes virtually impossible. Example: names of streets are stored in the same field ("street") like street identification numbers, depending on what is available. Data exchange based on a model that captures this information separately will most probably not be possible.

5.3 Are semantic conflicts easy to solve?

No general answer exists to this question as the range of conflicts is too broad. They differ in gravity, scope and resolvability. An example of an easy-to-solve conflict would be differing identifiers (names/tokens). These will just have to be renamed. At the other end of the scale of complexity, incompatible semantic concepts of the data in question render a solution to the conflict virtually impossible. The following list of tasks should give a good impression of the different severities of possible semantic conflicts.

5.4 Basic provisions against semantic conflicts

These issues must be considered in most cases of data exchange:

- The naming of elements may never be misleading.
- Data models must be described exhaustively. Even if the name of an element is appropriate for it to be understood correctly at first, communication partners might still misinterpret it. Only with a complete documentation can semantic conflicts between models be detected at an early

stage: Both the precise meaning and the data types of elements must be specified in their entirety.

- Multiplicities of elements must be indicated wherever they exist.
- The model must be semantically unequivocal. For any field, there should always be exactly one unit of information. A field should not be used for more than one semantic type of content (e.g. street name or index number) and the same content should not be attributable to more than one field.
- You should look into the level of granularity required for your (future) communication partners.
- Whenever a data model is changed because of diverse levels of granularity of the communication partners, it is usually rather easy to consolidate inventory data from a higher to a lower level of granularity. Before opting for the reverse direction, you should examine whether higher granularity is actually necessary. Steering clear of too high a degree of granularity can save you migration costs and reduce the risk of errors.

5.5 Avoiding semantic conflicts: Tasks

This is a list of tasks you should take into account when preparing the exchange of data. It lists frequently met semantic conflicts as well as strategies to avoid them or to handle imminent conflicts. Like the entire Asset Development Assistant, this section can be used as a check list of issues and tasks.
For any of the following tasks, you can send a specific question to the SEMIC.EU team by clicking on the question mark.

5.5.1 Describe completely

Data models must be described completely

- Even if the name of an element is sufficient for the modellers to understand, it might still be misinterpreted by communication partners. Semantic conflicts between models can be detected early-on when documentation is exhaustive
- In addition to the semantic specification, the data models of elements must be described completely.
- Multiplicities of elements must be indicated in their entirety

Conflict:
Elements of two data models have the same name but are used with different meanings.

Examples:

- The element "price" can use the same name for prices inclusive or exclusive of sales tax.
- An element "title" can contain values like "Mister", "Mistress" and "Professor", but in another model, the title can refer to a headline or name of a publication ("Tales of Mystery and Imagination").

5.5.2 *Name accurately*

The names of elements may never be misleading, even where the documentation is extensive.
Example:

- In a model, there is a "city" of the type "code" and also a "city abroad" of the type "text". The model is built on the assumption that the code list used for it only covers domestic locations and uses the field "city abroad" for any city which is not part of the code list. However, in case of reorganisations of local governments, there might also be (newly established) domestic cities not covered by the code list. Therefore, the field should not be named "city abroad" but, more accurately, "city uncoded".

5.5.3 *Provide for semantic inambiguity*

The model must not be semantically ambiguous: To any field, exactly one unit of information should be attributed. A field should never be used for more than one semantic piece of content (e.g. street name or identification number), and multiple fields should never be used for the same semantic content.

Risk:
In case of ambiguity, exchanging data with partners who possess an unambiguous model becomes impossible.

Examples:

- In a field "street", names of streets are stored along with street identification numbers, depending on what is available. Data exchange based on a model that captures this information separately will most probably not be possible.
- If the semantically same information in a model is modelled both as an optional object "Number" and as an optional object "RoleID" at a different point, it cannot be predicted where the information is represented in the concrete case of data exchange. In the worst case, there are different values in both fields.

5.5.4 *Consider requirements of future communication partners*

You should examine early-on which level of granularity current and future communication partners require.

Risk:
If data are migrated because of different levels of granularity of the communication partners' data models, there is a risk that a future communication partner requires the model that the data was migrated from.

5.5.5 Double-check the necessity of diverse levels of granularity

The necessity to preserve the portioning of data with different levels of accuracy in participating systems should be thoroughly considered.

Example of a conflict:

- There are two models containing the text element "family name". In one of the models, name prefixes like attributes of nobility are considered as a part of the family name, in another model there is a separate field for that purpose.

In order to rule out integration conflicts due to different levels of granularity, the technical and context-dependent necessity of different granularities should be examined. The costs of migration are a fundamental argument in finding a decision. If a higher level of granularity is not necessary for your purpose, a migration towards a coarser solution can prove easier to implement. This is, however, not the case where automatable rules for the integration are not applicable. Automatable rules for the separation of values are usually even more difficult to find.

Example:

A name prefix (e.g. "de") can easily be conflated with a name as the prefix is simply put in front of the suffix. In contrast, a separation can be challenging to decide on even for experienced staff. Further inquiries with the person in question become necessary (e.g. where it is unclear whether a part of a name is an attribute of nobility or a double name or in case of unfamiliar foreign names).

5.5.6 Primary keys: Determine and mark data fields

Task: Determining the specific information which identifies data sets unambiguously if this requirement exists. These data fields will then have to be marked as primary keys.
Risk: If data fields that are supposed to serve the unambiguous identification of data sets are not marked as primary keys, duplicate entries may emerge. Example: When an insurance number can by definition be used as an identifier of data sets, this distinctive feature must also be regarded technically since otherwise more than one data set can be created with the same insurance number (i.e. duplicate entries).

5.5.7 Identifiers: Define persistent unique IDs

Conflicting identifiers appear wherever attributes or entities are semantically equivalent but are still differently named. E.g. "identifier" and "id" for an attribute used to uniquely identify an object. Another possibility for conflicting identifiers is the use of the same identifier for attributes with different semantics.
In pan-European context this type of conflict will appear almost always if native language identifiers are used. Fortunately this type of conflict is quite easy to solve by renaming the identifiers in conflict. The hard part is to ensure that the attributes or entities in question are really and completely equivalent in their semantics. Task: For an unambiguous identification of data sets, persistent features must be

used or attributed.

Risk:
If unequivocal features of an object are used as an identifier which might change over time, there is a risk that data sets are created redundantly or are not found. Example: If car licence plates, e-mail addresses or combinations of name and address are used for identification, relocation can lead to the data sets not being found and the same person being represented by two different data sets.

5.5.8 Identifiers: Generate database-independent IDs

For data exchange between systems internal (data base specific) identifiers should not be used. Instead, additional identifiers must be generated which are independent from the used data base and which are also suitable for identification in other data bases, e.g. additional numbers counted up for every new data set.

Conflict: If the natural and unalterable features of an object (e.g. birth name, birthday) are not sufficient for unequivocal identification, IDs are assigned to the objects. Data bases attribute a single internal ID to every data set. If these IDs are also used by the business application for identification, a change of the data base can cause all identifiers to change. For an exchange of data with other data bases, therefore these internal IDs are not qualified.

5.5.9 Identifiers: Assign sole responsibility for the generation of IDs

When identifiers are generated, it must be specified which of the two systems involved in the data exchange is responsible for the attribution of identifiers.

If two systems attribute IDs independently, it cannot be warranted in the data exchange that same objects also have same IDs. It can only be ensured within a closed system that two objects do never get the same identifier and an object receives precisely one identifier.

5.5.10 Character sets: Determination

A character set (coding and the subset of allowed characters) must be determined for an asset.

5.5.11 Character sets: Harmonise the representation symbols

When systems are interconnected, it must be determined whether the same character set (coding and the subset of allowed characters) is used. Even if the same character set is used, it must be double-checked whether both systems process displayable characters coherently.

5.5.12 Character sets: Workaround for different character sets

If the lot of allowed characters in data exchange varies between communication partners and characters are used which belong to the mismatching portion, or if the depiction of non-representable characters is dealt with differently and ambiguously, it must at the least be determined how this challenge is tackled in practice. In many cases, only a rough approximation to the ideal solution is possible.

Conflict: If systems exchange data which is built on differing character sets (character coding and allowed characters), interoperability is at risk. For a data exchange format, precisely one character set must be determined. The conversion between two character sets causes a loss of information if the subset of allowed and actually used characters differ. This ranges from a restriction on capital letters without umlauts to US-ASCII, ISO-Latin-1 and Unicode. There are numerous other codings and application-specific restrictions of the characters to be used. Example: If the name of a person is stored in one system as "René Böttcher" and in another as "RENE BOETTCHER", the two systems are not interoperable. There is no bijective representation (one-on-one in both directions), because a "Rene Boettcher" from the first of the two systems would also be stored as "RENE BOETTCHER" in the other, exactly like "René Böttcher".

5.5.13 Character sets: Ensure that restrictions can be validated

If it is impossible to determine a full character set like Unicode for a data exchange standard, you should select a subset which can easily be defined in XML Schema. This will facilitate automatic checking of the restriction, e.g. ISO-Latin-1 (it can be sensible, however, not to resort to ISO-Latin-1 directly if this spares code changes because internally - outside the data exchange format - a larger set of characters can be used).

Risk:
If not all characters from a character set like UTF-8 can be applied, restrictions are needed. In XML Schema, a detailed listing of allowed characters can hamper the performance in validating data sets (i.e. the automated checking of compliance with restrictions).

5.5.14 Unique transliterations between languages

For the acquisition of data and the search for data, precise rules for transliteration must be defined. Transliteration is the transposition of characters of the original script to the characters of the script supported by the IT system. Where, for instance, only Latin characters are supported, any text in Greek, Bulgarian, Russian, Chinese, Japanese, Korean, Arabic etc. must be transliterated. If not all diacritic letters of Latin script are supported, transliterations must also be defined for several other European languages (ä -> ae, é -> e, ø -> oe). In the data exchange between systems, the targeted lot of transliterations can only be determined out of those characters that are supported by all systems.

In many cases, data must be transferred from the original spelling in one character set to another supported by the IT system. If this task is carried out on an individual basis, this can lead to different representations of an original name already within a single IT system. Example: If the name of a

Russian person is registered, „Горбачёв“ can be transformed to "Gorbachev" (English phonetics), "Gorbatschow" (German phonetics) or "Gorbačëv“ (scientific transliteration). Duplicate data sets can result and data entries might be missed during a research.

5.5.15 Bijective transliterations between languages

If data sets must be attributable to their original spelling after a transliteration, the rules for transliterations must be bijective, i.e. uniquely transvertible in both directions. In most cases, this requires support by diacritic characters by all participating systems.

Example:

"Горбачёв" <-> "Gorbačëv"

A bijective transliteration without diacritic characters is rather unhandy.

Example:

"Möller -> "M"oller" or "René" -> "Ren´e"

You must be aware that after the transliteration of data, data sets which were originally different might not be distinguishable anymore.

Example: A system registers surnames with diacritic characters ("Moeller", "Möller", "Møller") and another system replaces these characters uniquely but not invertible (both as "Moeller"). This makes the identification of data sets in the data exchange between the systems considerably more difficult.

5.5.16 Data types: Define conversion function

Conflicting data types frequently appear in data exchange scenarios. One example are numbers that can also be stored as character strings. Sometimes this is even necessary, e.g. when storing Latin figures.
Find a function that allows transferring from one data type to the other. If you succeed, this is a rather easy and swift solution to this type of conflict. These functions can however entail a loss of precision, e.g. by converting from double to integer. It is important to clear the semantics and document any potential loss of data or precision.

5.5.17 Default values: Identify conflicts

Default values are often used in data sources, e.g. to signify unknown data elements. Using a date like 01/01/1970 is an example. If these default values are not adequately documented and/or different this can lead to misinterpreted data. It is therefore important to identify any default values used and agree on the meaning of any default values used.

5.5.18 Integrity constraints: Find common semantics

This type of conflict is closely related to data base design. It occurs when data from two different sources is to be integrated but is subject to different integrity constraints. Using different values as identifier keys is one example.
Such conflicts usually point to a problem with semantic equivalence as well.

5.5.19 Scaling and units: Identify and harmonise

Whenever an amount is stored, it is important to be aware of the unit and scale this value is used with. E.g. German brokers give the values of their stocks in Euro with two decimal places. British brokers give values in Pence with no decimals.
Even if two numbers share the same semantics, they do not always add up. Since everybody has his own background we tend to naturally assume numbers to have specific values and scales.
When a bridge over the river Rhine was built, starting simultaneously from the German and Swiss side, both sides of the bridge met with a different height. It was previously agreed to measure all heights as meters above sea level. But the interpretation of "sea level" differs by 25 cm as Germany uses the mean sea level at Amsterdam while Switzerland uses the sea level at Marseille.
These conflicts are easy to spot but equally easy to miss, as practice shows.

5.5.20 Scaling and units: Double-check correct use

The gravity of errors based on incorrect units and disparate scales makes testing indispensable. Prepare a set of test data and check for the accuracy of the exchanged data.

5.5.21 Scaling and units: Use appropriate conversion functions

If it is indispensable to maintain different units or scales of values, you must make sure that conversion is provided for accurately.

5.5.22 Data representations: Find agreements or conversion functions

A very common occurrence of this type of conflict is in date representation. A date given as 08/09/07 could be interpreted as 9 August 2007, 8 September 2007, or even 7 September 2008.
Wherever the representation of data is not given by using a certain data type, you should look out for this type of conflict, decide on conventions for the representation of data and document all these decisions.

5.5.23 Aggregation: Find a common level

Data models are often radically different. This is partly due to differing requirements but to a large extent also due to personal preferences of the person modelling the data concepts. Especially the amount of detail a single concept contains varies greatly. This leads to different levels of aggregation

and the conflicts arising from it.
When trying to reconcile such conflicts you have to keep in mind, that it is fairly easy to transform data from a very detailed model to a higher aggregation. Breaking up integrated data for use in more detailed models is usually very hard.

5.5.24 Semantic concepts: Negotiate and define a common basis

If semantic concepts are really incompatible it is virtually impossible to reconcile such a conflict. Unfortunately, due to different legal, cultural, political, and social backgrounds, incompatible semantic concepts must be expected to appear in pan-European contexts.
In case of such a conflict it will be necessary to formulate new common concepts. This will typically require face-to-face negotiation between all parties concerned.

5.5.25 Missing data: Define a scenario

Whenever exchanging data between different partners it should be clear beforehand what will happen in case some data expected by one party cannot be provided by the other party. In pan-European context this type of conflict will frequently arise, since data is collected subject to different legal regulations and different data protection laws.
You should ensure that all partners in a data exchange know how to handle missing data.
All participating parties must agree on means to handle missing data on the application layer.
If default values are used to handle missing data the interpretation of such values must be unambiguous. It is usually not advisable to use default values that could also appear as regular data, e.g. using 1 January 1970 as a default date.

5.5.26 Redundant data: Identify and define a scenario

In data base design and even in everyday life, we encounter redundant data. E.g. the town given in a postal address is often redundant with the postal code as is the area prefix of the phone number. It is advisable to document any redundant encoding of semantic concepts.
Another conflict can arise when integrating data from multiple sources. In these cases the same data may exists in multiple objects. These "doubles" do not always have to be apparent. Your solution should be able to handle such uncertainties.

- Try to avoid redundant data
- If impossible to avoid, agree on means to handle on application layer

5.5.27 Semantic Statements: Documenting the solution of Semantic Conflicts

It is highly recommended for all encountered conflicts to document the problem, the solution and its reasoning thoroughly. This will not only highlight potential future conflicts (if the scope and quality of the data change) but will also help others to avoid similar conflicts (especially in cases of reuse of

existing interoperability assets).

Method:
Your asset should contain "semantic statements" that state the exact agreed semantics of each attribute and object

Add a "semantic statement" to each attribute and object. The statement must contain the exact agreed semantics. In most cases it will use natural language to describe the exact meaning of every attribute and object.

6 Prototype

6.1 Document the prototype

6.1.1 Prepare test data

The use and reuse of any data exchange model is greatly eased by providing adequate test data. This test data should also be made a part of the final asset uploaded to SEMIC.EU. Thus users of your asset will be better able to understand your asset and implement it correctly.

6.1.2 Set quality criteria

As the owner and developer of the asset, you should define criteria against which your solution can be proofed (using the test data).

6.2 Implement Asset and Upload to SEMIC.EU

6.2.1 Document your implementation

For an asset to enter the clearing process on SEMIC.EU, a comprehensive documentation is required. As described above, this will make your solution easier to maintain and extend.

6.2.2 Pack and upload your asset to SEMIC.EU

Pack a zip file containing:
- all artefacts relevant to your asset (e.g. XML schema, UML-models, etc.)
- documentation, preferably in English
- test cases and test data (if available)
- the licence file

6.2.3 Leave your feedback for the SEMIC.EU community

Especially if you have reused an asset from SEMIC.EU your feedback will be useful for any future user of this asset. So please leave your feedback detailing how you were able to use the asset, what challenges you met and what problems had to be solved. You are also welcome to suggest improvements to the asset you reused.