How to Solve the Business Standards Dilemma

CCTS Key Model Concepts

Disclaimer & Liability Notice

How to Solve the Business Standards Dilemma – CCTS Key Model Concepts

Applies To:

SAP NetWeaver.

Summary

The first article in the series "How to Solve the Business Standards Dilemma" discussed a new approach for semantic data modeling as defined in *ISO 15000-5 Core Components Technical Specification.* The CCTS approach is a paradigm shift in defining, describing, or representing business data to enable real application, platform, or B2B interoperability. Many readers recognized that many of the key concepts in that specification have been around since the beginning of electronic data, and they wonder if we can really achieve interoperability with these existing approaches. The quick answer is yes, because CCTS combines these traditional approaches with new concepts made possible by new technologies. Specifically, the ISO 15000-5 specification provides for:

- logical and semantically driven modeling methodology for getting consistent and unambiguous understanding of data models
- context driven and collaborative framework for evolutionary modeling, modification and usage of reusable artifacts, and a
- syntax independency so that data models can be transformed in different physical syntaxes to always enable the same semantic-based understanding.

However, in order to get a deeper understanding, additional articles that provide greater detail are necessary. These articles will assist the reader in developing a deeper understanding and appreciation for CCTS, and will identify why the CCTS approach is especially helpful in achieving true data interoperability in the B2B world.

Process and data modeling are as much an art form as a science. The current freedom in modeling methodology leads inevitably to different complexities, with different cardinalities and different structures for identical entities in data models and within the structure of a business message interface. The result of this means always a very complex and expensive mapping. Additionally, the most of current modeling languages, like UML or XML Schema Definition Language, focuses on the technical representation and implementation. They do not say how business entities must be named, typed, positioned and assembled in logical correct order so that the semantic of a structure could be understood unambiguously.

A logical and semantic based modeling of objects is comparable to a logical composition of sentences, paragraphs, business letters, articles or even books. It helps to assembly reusable and understandable structure into a business data model on semantic level. It provides also the highest possible freedom to define the semantics of business information for every requirement.

The second article “CCTS Key Model Concept” describes the key aspects and representation of the data model as described in the ISO / UN/CEFACT Core Components Technical Specification "CCTS". It considers especially the template realm of data models and the derived context specific realm of a data model, which will represent the precise semantic meaning in a given business content. This kind of data model concept helps to enable a business data exchange with less bilateral arrangements beforehand. The focus of that business data models is on similar naming, structuring and assembling so that the involved business partners can interpret and use these business data models in the same way without any ambiguity.

By: Gunther Stuhec

Company: SAP AG

How to Solve the Business Standards Dilemma – CCTS Key Model Concepts

Date: 02. March 2006

Table of Contents

Introduction

Beyond traditional syntax rules and modeling languages, the primary concept of the Core Components Technical Specification (CCTS) is its building block system that enables reusable and common understandable data. The CCTS follows key concepts of conceptual, logical and physical data modeling,

reflects Codd's rules[1] and normalization for data base management systems, and aligns with the current OO-modeling approach as typified by UML[2] from OMG. Beyond this, CCTS has very important aspects for enabling the common understanding and reuse of data, through semantics and context.

Overview of the Concept

Before we go in more detail of the CCTS modeling approach, you must first understand the key concepts of CCTS as shown in figure 1. These key concepts cover two focus areas – the conceptual Core Components (CCs) and the physical/logical Business Information Entities (BIEs).

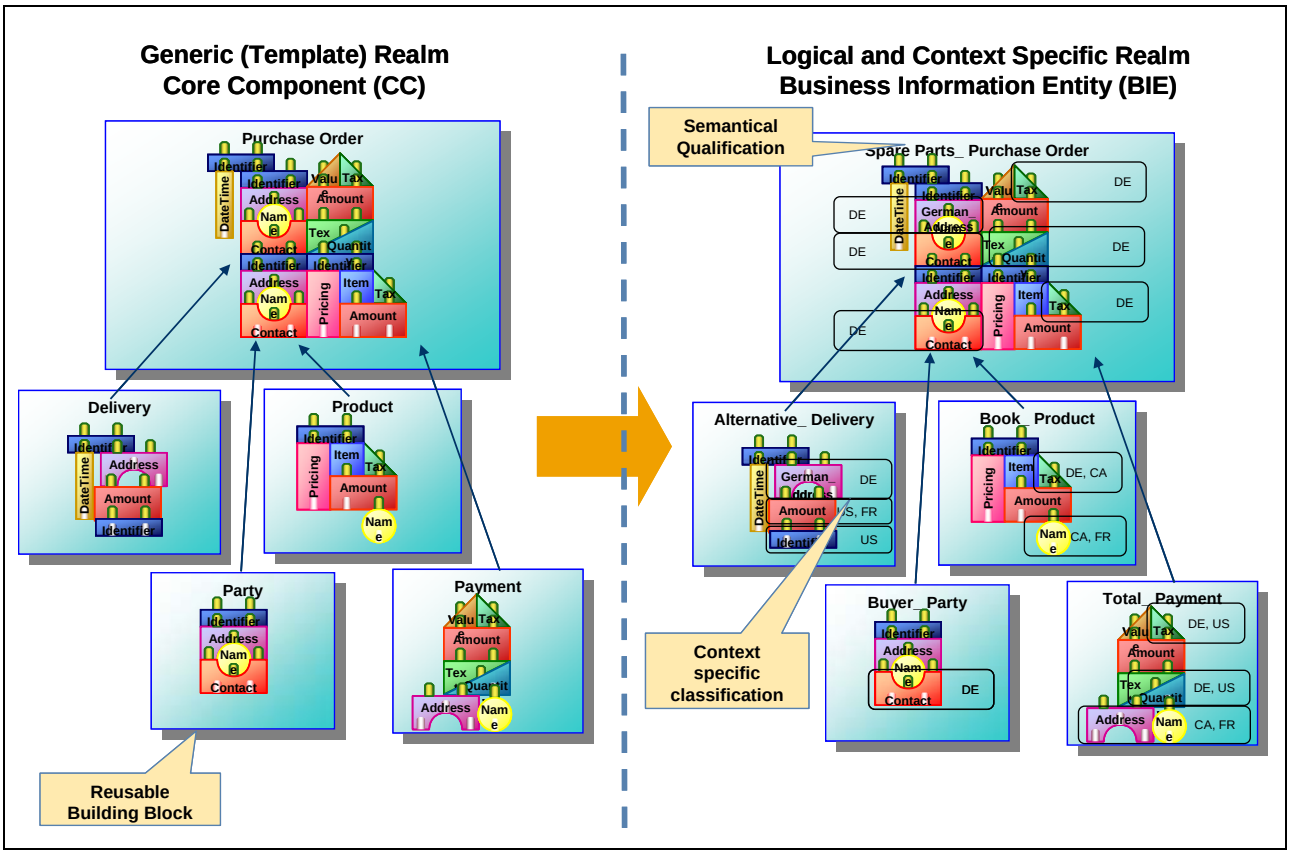


Figure 1 - Template and Context

Core Components are building blocks with generic business semantics and purpose. CCs provide a context independent template that are used for developing components in a given business context with semantically

---

[1] Codd's rules at Wikipedia free encyclopedia: http://en.wikipedia.org/wiki/Codd%27s_12_rules

Codd's rules at SQL Server Central: http://www.sqlservercentral.com/columnists/fkalis/coddsrules.asp

[2] UML Resource Page at OMG: http://www.uml.org/

UMLWiki: http://umlwiki.org/index.php?title=Main_Page

and logically correct structure and content. Once the business contexts are identified, the CCs can be transformed to reflect context. The results of this context-based transformation are BIEs that constitute real-world physical/logical data constructs. Since all BIEs are based on the same context-neutral conceptual data models, the context-neutral/context-specific transformation precludes independent developers from creating very different BIE structures for the same semantic concept of a complete assembly.

We detail these key CCTS principles in the remainder of this article.

Conceptual Model Templates

CCTS conceptual model template constructs are the focal point in the definition of unambiguous business data models. These templates play a pivotal role in transforming context-neutral components into contextualized components. These templates provide a mechanism through building block derivation by restriction for flexibility in the required business contexts. [3]

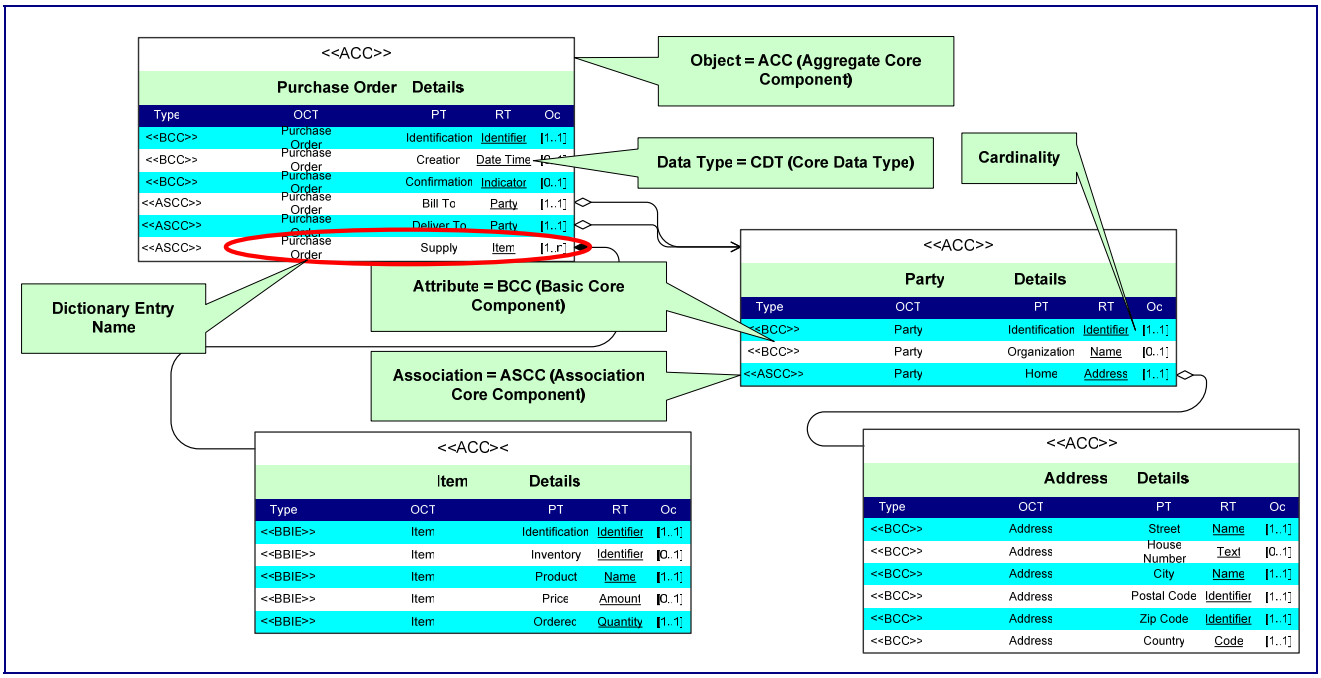


Figure 2 - Core Components as Generic Templates in the Conceptual Realm

The contextual components function as reusable building blocks for flexible data models and business messages. Reusable building blocks and flexible data models and messages are an important innovation. In this approach, the business information is semantically concise and thus easily interpreted and universally understood by anyone and everyone. The result of these templates is to accomplish what would otherwise be considered mutually exclusive objectives – flexibility and autonomy for responsive industry solutions and cross industry interoperability, all with semantically unambiguous data bases and business messages.

[3] Derivation by restriction will be explained in the forthcoming article "The Semantic Precise Entities of the Context Specific Realm"

The idea behind this principle is that all business data follow similar semantic concepts. specifically, the naming and structuring of the Core Component building blocks (see figure 2) will consistently occur regardless of the developer. To achieve this, the Core Components must have:

- A common and generic modeling concept for objects and data
- A naming convention for definition of the generic semantic meaning in the Dictionary Entry Names
- A fixed set of reusable data types for consistent business value representation

One of the elementary key concepts inherent in the generic and reusable representations of CCTS constructs is their alignment with UML object class concepts. Core Components are in fact specific representations of information in UML object classes for the conceptual model. Higher level Core Components are self-contained entities that are described by their generic and reusable characteristics just as UML classes are. Core Components may establish associations to other object classes just as UML classes can. Since these Core Components are intended to be contextualized to describe the specific business environment in which the Core Component must be used[4], they should describe real things that can easily be identified and used for business reasons. In other words, a Core Component should represent real-world objects, which can be used in every context such as – "*car*", "*party*", "*person*", "*location*", "*credit card*", "*amount*", "*text*" etc. Core Component objects may or may not be used in different industries, countries and/or business processes.

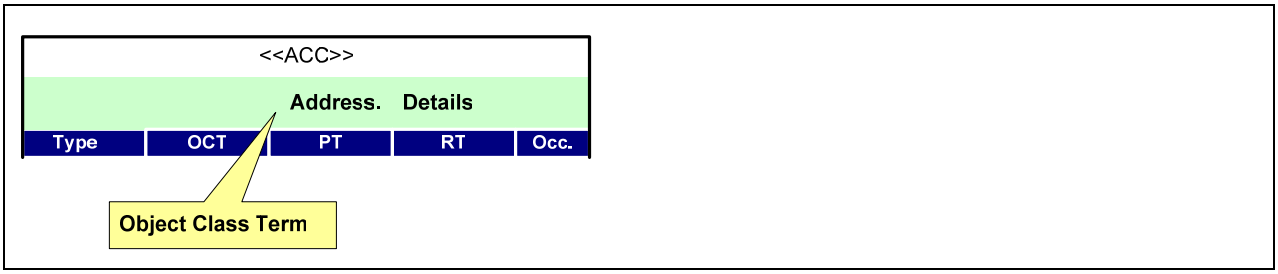
There are three types of Core Components:

- **ACC** – Aggregate Core Component
- **BCC** – Basic Core Component
- **ASCC** – Association Core Component

Each of these constructs will be explained in more detail in the following subchapters:

ACC – Aggregate Core Component

An Aggregate Core Component is a collection of related pieces of business information that together convey a distinct real-world object with a specific business meaning, independent of any specific business context. Ideally an Aggregate Core Component should be "normal", the data modeling world's version of cohesive. A normal Aggregate Core Component depicts one concept, just like a cohesive class model represents one concept. For example, the concepts of Party, Purchase Order, and Address are clearly three different concepts, and each is modeled as a separate Aggregate Core Component. In CCTS,



[4] The context driver principle and its behavior will be explained in the SDN article "How to Solve the Business Standards Dilemma – The Context Driver Principle".

Figure 3 – Construct of an Aggregate Core Component

The name of the real-world object will be shown by the "Object Class Term". Additionally, a suffix ("*Details*") is appended to each class to reflect that the ACC contains all relevant and generic properties of this real-world object. So for the classes of Person, Purchase Order, and Address, the ACCs would be – "*Person. Details*"[5], "*Purchase Order. Details*", and "*Address. Details*". Each of these ACCs includes Basic Core Components and Association Core Components that are relevant for expressing the properties of the class.

BCC – Basic Core Component

An Aggregate Core Component has at least one, and possibly more, Basic Core Components. These BCCs constitute a singular business characteristic of that specific Aggregate Core Component. Every BCC describes a single property, which can be used in any business context. Figure 4 shows an example BCC "Address. Postal Code. Identifier".

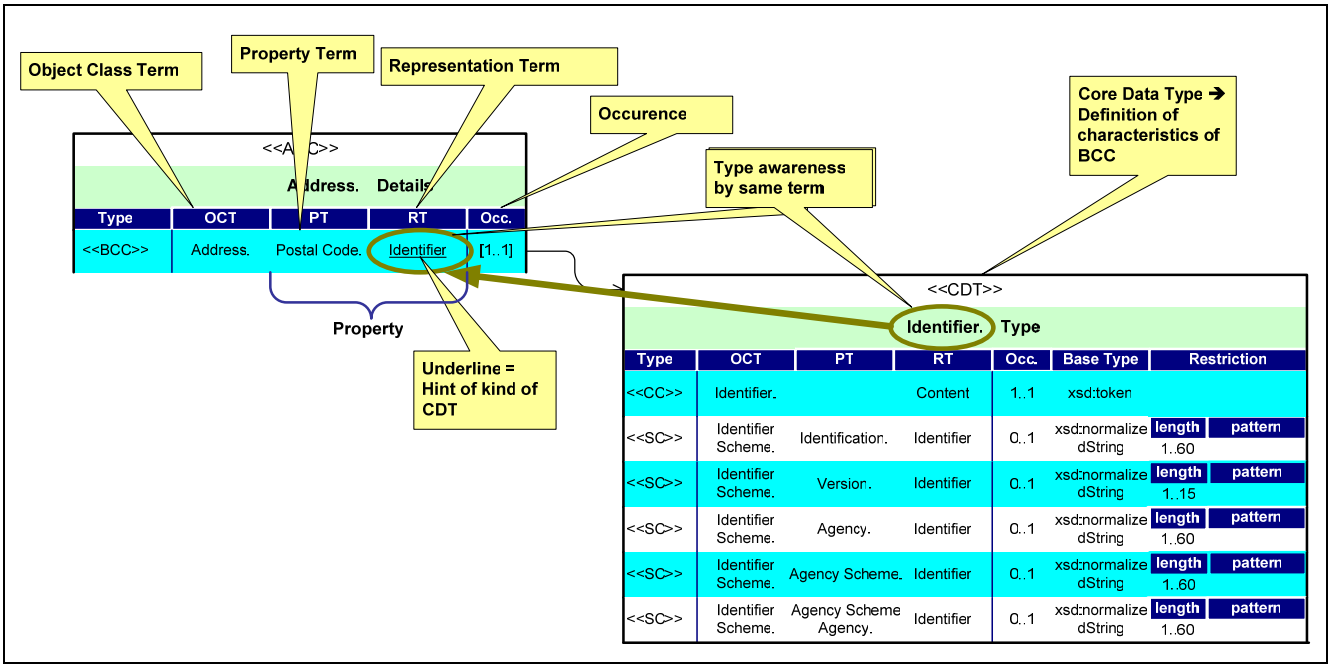


Figure 4 – Construct of a Basic Core Component

The Dictionary Entry Name (DEN) of a Basic Core Component (BCC) is unique and concise. The DEN consists of an Object Class Term (OCT), a property term (PT), and a representation term (RT). An example of a complete Dictionary Entry Name of a BCC is "Address. Postal Code. Identifier".

The Object Class Term represents the object class for which the BCC is defined, for e.g. "*Address*". The Property Term is the first part of the name of an administered item. The Property Term expresses a property of an object class, and defines the characteristics that are belonging to an object class. The Representation Term is the second part of the property. It defines the subject that is used to express the kind of

[5] All naming conventions of Core Components and Business Information Entities will be described in the SDN article "How to Solve the Business Standards Dilemma – The Naming Conventions."

representation of the property and reflects it's datatype. For e.g. "*Postal Code*" gives the apparent proposition and the "*Identifier*" says how the "*Postal Code*" will be represented. A BCC always gets its characteristics for the representation of the values by an appropriate Core Data Type.[6] Type awareness of the CCs is easily determined by examining the BCC Representation Term of the BCC, as a Representation Term is always the same term as the Object Class Term of its corresponding Core Data Type.

It is also possible to define the occurrence of a BCC. The occurrence (Occ.) expresses the maximum number of instances of the BCC that a single instance of an entity is permitted to have. The occurrence is represented as follows:

- **1:1** – The occurrence of one specific BCC in the instance is required.
- **1:n** – The occurrence of the BCC in the instance is required, but the maximum number of instances of this BCC is infinite. It is also allowed to restrict the maximum number of occurrences by an integer value.
- **0:1** – The occurrence of one specific BCC in the instance is optional.
- **0:n** – The occurrence of the BCC in the instance is optional, but the maximum number of instances of this BCC is infinite. It is also allowed to restrict the maximum number of occurrences by an integer value.

ASCC – Association Core Component

Sometimes the content model of an Aggregate Core Component consists of other Aggregate Core Components. For example, a person has an address, a delivery shipment contains one or more packages, and so on. These sub Aggregate Core Components constitutes further complex business characteristics of the parent Aggregate Core Component. Therefore an association must be defined between these Aggregate Core Components. In CCTS this is accomplished by creating what are called Association Core Components. As shown in figure 5, the Association Core Component represents association between the associating ACC *Person. Details* and the associated ACC *Address. Details*.

[6] The construction of the CDT will be described in the article "CDT- Core Data Types".

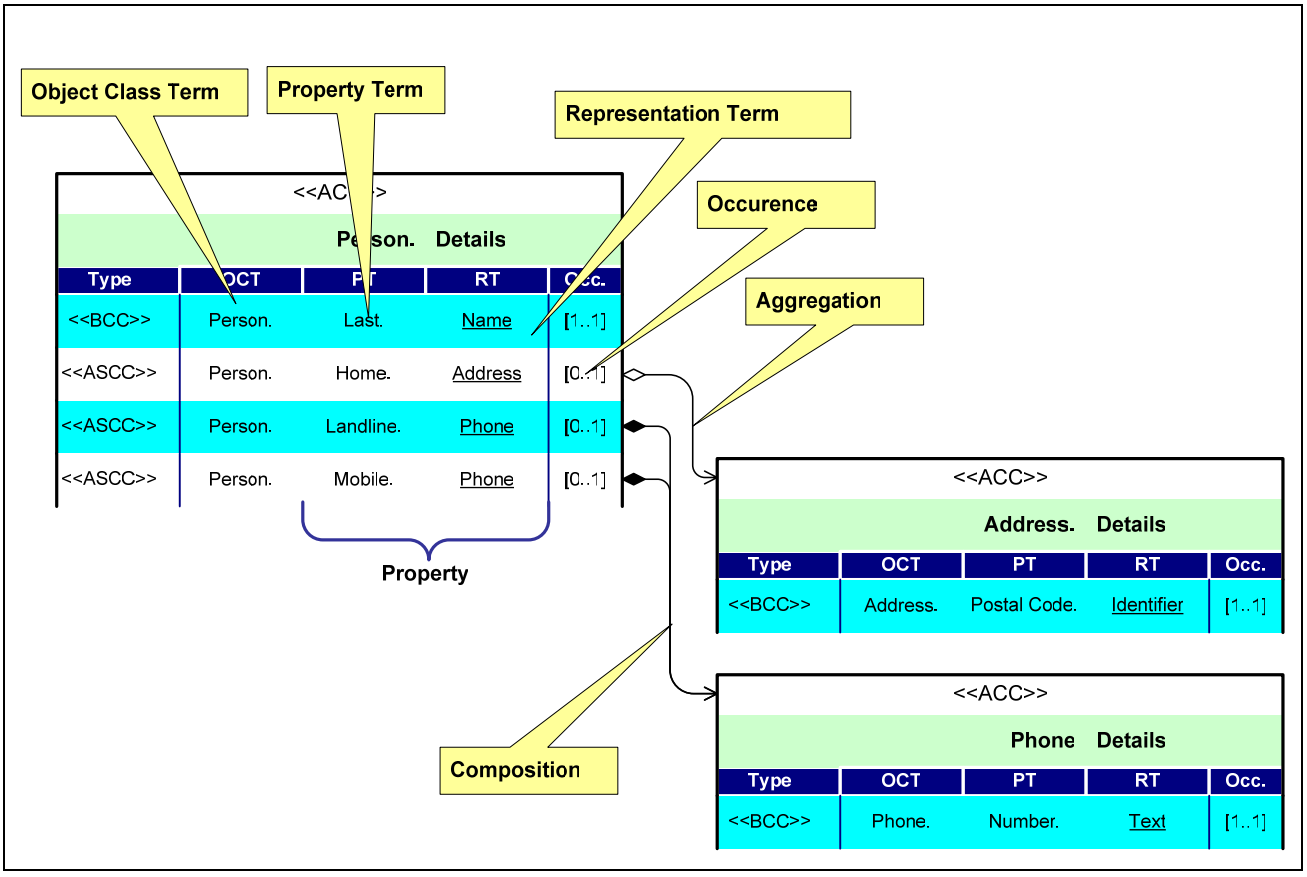


Figure 5 – Construct of Associaton Core Components

The construct and behaviour of the Dictionary Entry Name (DEN) of an Association Core Component (ASCC) is the same as the Dictionary Entry Name of a BCC. It is a unique and concise name that consists of an Object Class Term (OCT), a property term (PT), and a representation term (RT). An example of a complete DEN of an ASCC is "*Person. Home. Address*".

The only difference is the Representation Term (RT). Instead referring to a CDT, the RT of an ACC represents another ACC to which an association relationship is required. This associated ACC is a sub aggregation that describes an object class with a complex structure, which is required for the specific ACC. For example, to complete the ACC "*Person. Details*", a complete structure of a address is also necessary. This address is described by the ACC "*Address. Details*". The association from the ACC "*Person. Details*" to the ACC "*Address. Details*" causes the content model of the *Address. Details* ACC to be included in the content model of the associating ACC *Person. Details*. It is also necessary to express the nature of the association, or in other words, how the "*Address*" will be used as a property of the Person. Details ACC. This information is conveyed as the property term of the ASCC DEN. For our example, "*Home*", "*Business*", "*Holiday*" etc. might apply.

All associations which are expressed by an ASCC are unidirectional. By this we mean that one ACC (e.g. Partner. Details) knows about the other ACC (e.g. Address. Details) and the relationship, but the other ACC (e.g. Address. Details) does not. This is reflected in UML by an open arrowhead to point to the ACC that is known. Only the associating ACC reflects the association as part of its content model.

Two types of associations are allowed in CCTS. These are:

- Association by **aggregation** – the associated ACC is a part of the another ACC. The associated ACC (e.g. Address. Details)that forms a part of the associating ACC (e.g. Partner. Details) functions both as an ASCC but can also exist independently. Thus the existence of the associated ACC is not determined by the associating ACC. In UML, the aggregation association is represented using an empty diamond symbol on the association line between the classes, and appears adjacent to the associating class.

- Association by **composition -** The associated ACC only occurs as part of the associating ACC, never independently. Thus the existence of the associated ACC is determined by the associating ACC. In UML, the composition association is represented using a filled diamond symbol on the association line between the classes, and appears adjacent to the associating class.

Once the type of association – aggregation or composition – is set for a specific ASCC, it is not changeable. Figure 5 shows that two ASCCs "*Person. Landline. Phone*" and "*Person. Landline. Phone*" based on the same ACC "Phone. Details". The type of association of both ASCCs is a composition. The type of association must always be identified to ensure proper expressions of the resultant context specific Association Business Information Entities and syntax specific expressions for information exchange.

CDT – Core Data Types

A Basic Core Component is not sufficiently precise enough to function as a piece of data. That is because the BCC is a high-level semantic representation of data without detailed data type information. For example, the BCC "*Item. Price. Amount*" tells us that the underlying Data Type is an amount, but it gives us no clue as to the details of the data type – such as its primitive, any content restrictions, or supplementary information. To capture that information, CCTS has defined Core Data Types (CDTs).[7] Every BCC must be based on a CDT, as reflected by its representation term. The Core Data Type itself represents the smallest piece of information in a business data model. Since Core Data Types are intended to be universal in their use by the universe of business data, they have no business meaning themselves. As we have learned, that business meaning is conveyed by the using BCC. For example, the BCC "*Item. Price. Amount*" conveys business meaning, whereas it's Core Data Type of "*Amount. Type*" (See figure 6) has none.

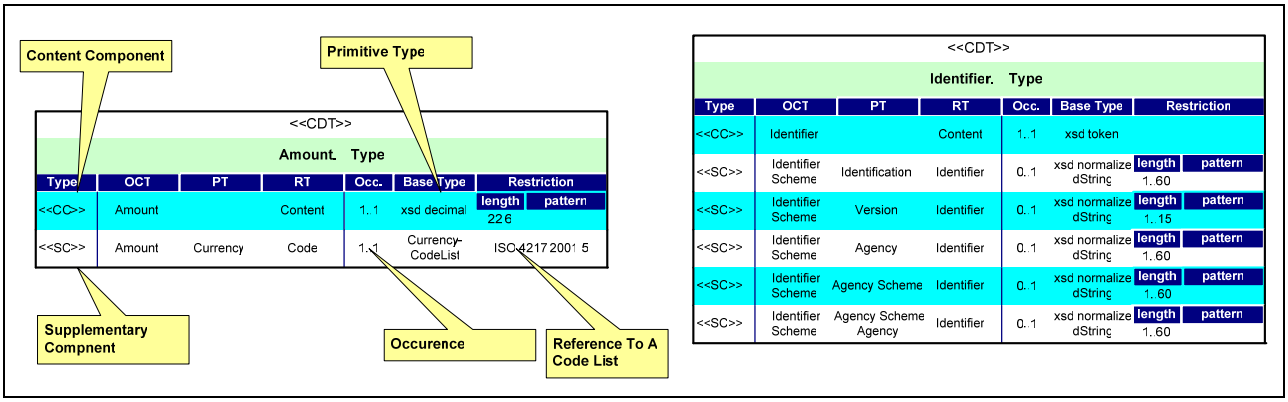


Figure 6 - Construct of Core Data Type

[7] CCTS version 2.01 does not explicitly present the Core Data Types. Rather they are inferred through the Core Component Types and allowed Primary and Secondary Representation Terms. The forthcoming Version 3.0 of CCTS addresses this confusion by specifically defining the listed Core Data Types and all associated content and supplementary component information for each of them.

A Core Data Type defines the nature of the content of the BCC, and provides supplementary information that gives essential extra definition to the content. The content is expressed by the Content Component. Content Components are based on a primitive type for typing the data (e.g. decimal, integer, string etc.). Content can also be restricted by length or a proscribed pattern, which is normally identified by a regular expression. The supplementary information will be expressed be one ore more Supplementary Components. Supplementary Components might include the nature of the identification scheme or code list, the owner of the scheme, its Uniform Resource Identifier and so on. Such metadata provides additional information necessary for the understanding of this primary business information.

The Core Data Type of "*Amount. Type*" in figure 6 defines an amount with a corresponding currency unit. The content component carries the decimal representation of the amount value. This value has no meaning by itself as it is just a number. The additional supplementary component "*Amount. Currency. Code*" provides the additionally required information – currency code – that is necessary to understand what the amount is expressing in complete terms. In CCTS, the "*Amount. Currency. Code*" supplementary component is based on an international code list for currency codes – ISO 4217 – to ensure that the value of amount will be unambiguously interpreted by users and machines.[8]

The CCTS considers only a fixed list of Core Data Types. This list currently includes "Amount. Type", "Binary Object. Type", "Code. Type", "Date. Type", "Date Time. Type", "Duration. Type", "Graphic. Type", "Identifier. Type", "Indicator. Type", "Measure. Type", "Name. Type", "Numeric. Type", "Percent. Type", "Picture. Type", "Quantity. Type", "Sound. Type", "Text. Type", "Time. Type", "Rate. Type", "Ratio. Type", "Value. Type", and "Video. Type".[9] These Core Data Types and how they should used will be explained in the SDN article "How to Solve the Business Standards Dilemma – Data Typing".

The Semantic Precise Entities of the Context Specific Realm

As we stated previously, Core Components are conceptual in nature. They never appear natively in physical/logical data models or business information exchanges. That is the domain of the context-specific Business Information Entities (BIEs).

[8] ISO 4217:2001 Codes for the representation of currencies and funds

[9] Additional Core Data Types may be defined by the CCTS team through inputs from users.

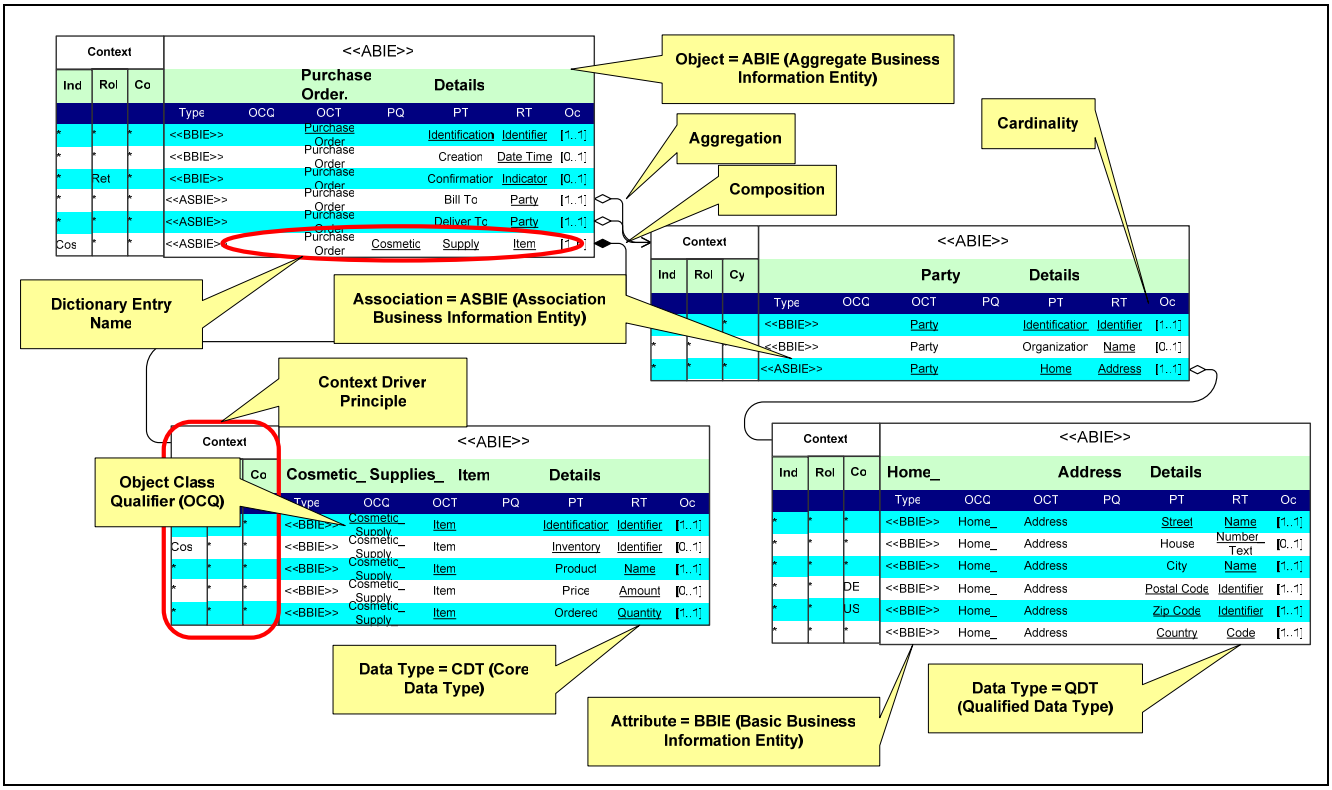


Figure 7 - Business Information Entities for Business Exchange in Context Specific Realm

In keeping with the tenants of CCTS, all BIEs must represent an unambiguous and interpretation free business semantic definition in a given business context. The holy grail of business information exchanges is for business data to be exchanged between business partners without prior agreements. This is a difficult proposition, and it is fair to say that is still not reached, but the CCTS metamodel with its use of strong semantics and context goes a long way towards achieving this goal. To avoid two independent developers ending up with very different BIE structures for the same semantic concept of the complete assembly, all BIEs are derived from a single Core Components model. To achieve this, physical/logical BIEs Must:

- have the same metamodel as the conceptual Core Components,
- always be derived from corresponding Core Components (use derivation by restriction mechanism),
- be of data types that are derived from the Core Data Types with restrictions specified in accordance with given business requirements, and
- explicitly state context using CCTS context methodology and agreed upon context values.

Figures 7 illustrates how BIEs follow the same key concept model as the Core Components, but are based on the derivation by restriction mechanism.

The derivation by restriction mechanism is very helpful in achieving naming and structuring that provides precise and unambiguous semantics. But what is the CCTS derivation by restriction mechanism? In fact, it is really a combination of the restriction of the content model, and restriction of the semantics through the application of additional qualifiers. The restriction of the BIE semantics through the application of qualifiers reflects restrictions of the template CC. These restrictions represent unique business requirements. Since

business requirements vary, multiple BIEs – each with their own content models – can be created. Each is semantically unique, and each reflects restriction on structure or characteristics. For example, we defined one BCC, the "*Address. Postal Code. Identifier*". This BCC could be used within the same aggregation in the context specific realm. It is possible – depending on the cardinality of the underlying BCC, that the Address. Details ABIE might contain two entities - "Address. Street_ Postal Code. Identifier", and "*Address. Post Office Box_ Postal Code. Identifier*". Both based on the same BCC and both used in the same aggregation.

The following sections describe:

- How the specific Core Component ACC, BCC, and ASCC constructs can be derived into the context specific realm
- The derivation by restriction mechanism in more detail for every artefact.

ABIE – Aggregate Business Information Entity

An Aggregate Business Information Entity is always derived from an ACC with the same generic business semantic meaning. However, the difference between and ACC and an ABIE is that an ABIE reflects the real-world object with a particular business meaning in a specific business context as expressed by the qualifiers and context values.

As shown in figure 8, an ABIE includes more precise semantics through additional qualification of the names, and semantically more precised properties conveying the real-world object in the specific business context. Each ABIE reflects a derivation by restriction from the source ACC.

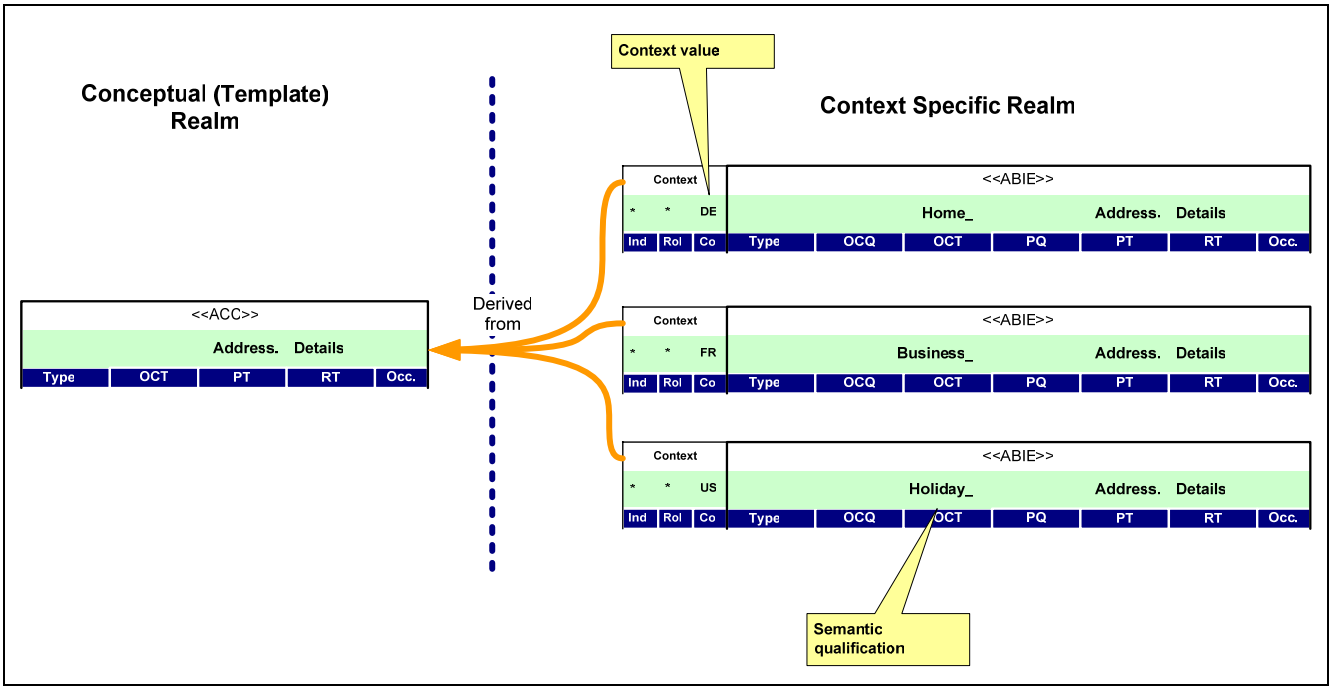


Figure 8 – Derivation and usage of ABIE

In our example, there are three derived ABIEs; each of which is semantically restricted and contextually specified from its template ACC "*Address. Details*".[10] The semantic restriction is defined by the semantic qualification and the contextual specification is expressed by the context values. Each of the properties of the underlying ACC can be restricted according to specific semantic qualification and/or specific context. That means, these ABIEs may have different BBIEs and ASBIEs, each tailored to the specific business requirements of the ABIE.

For example, the qualified ABIE "*Home_ Address. Details*" might have a different structure than the qualified ABIE "*Business_ Address*". Most of the entities inside these ABIEs are similar, but some of them fit only the specific business requirement for representing an address for a home, business, or holiday. The business address might have a specific postal code for companies whereas the home address would not.

Figure 8 shows further context categories, like "*Rol*" for Business Role and "*Ind*" for Industry. Our example ABIEs will have no restrictions in these context categories, and thus will have the CCTS assigned value of "in all contexts". These are expressed in our example by the wildcard sign (*).[11]

BBIE – Basic Business Information Entity

The Basic Business Information Entity (BBIE) represents a business meaningful property of the ABIE to which it belongs. The BBIE includes content and relevant supplementary information to this content in a given business context. As shown in figure 9, several BBIEs can be based on a single BCC. It is also possible that several BBIEs, each derived from the same BCC, can be defined in the same ABIE (e.g. Home_ Address. Details). In this approach, the source BCC is used to create several BBIEs within the same ABIE. Each of the BBIEs represent a restriction on their source BCC through semantics and possibly by their characteristics.[12]

---

[10] The CC model is derivation by restriction. Each ABIE must be a subset of its source ACC. However, the notion of extension is also present in CCTS in the sense that multiple ABIEs can be created from a single ACC. However, we don't refer to the CCTS methodology as derivation by extension so as to avoid confusion with developers who might believe that we can extend the ABIE content model to include properties not found in the source ACC.

[11] This is only a very rough description of the context driver principle. Context will be described in detail in one of the next SDN article in our series - "How to Solve the Business Standards Dilemma – The Context Driver Principle".

[12] The CC model is derivation by restriction. Each BBIE must be a subset of its source BCC. However, the notion of extension is also present in CCTS in the sense that multiple BBIEs can be created from a single BCC. However, we don't refer to the CCTS methodology as derivation by extension so as to avoid confusion with developers who might believe that we can extend the ABIE content model to include properties not found in the source ACC.

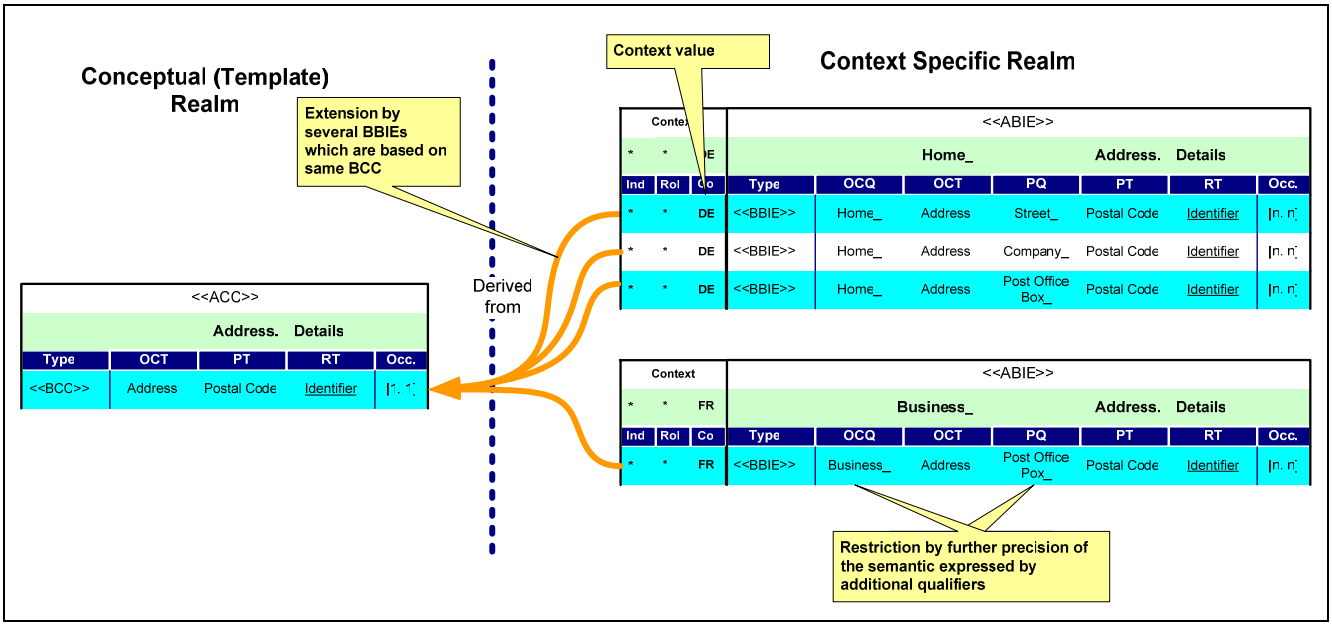


Figure 9 – Derivation and usage of BBIE

Following this approach, a BBIE could be the specific postal code for the street within the home address (DEN: *Home_ Address. Street_ Postal Code. Identifier*), or the specific postal code for the company within the same home address (DEN: *Home_ Address. Company_ Postal Code. Identifier*). Each has the same primary semantic of "*Address. Postal Code. Identifier*" that comes from the source BCC. The additional Property Qualifiers of "*Street_*" and "*Company_*" add more precise semantics to each BBIE.

These BBIEs may be unique to a home address that is only defined for Germany (Co = DE). Other addresses for other countries may have more or fewer postal codes. This depends on the specific business requirements for the given business context. For example, the "*Business_ Address. Details*" that is specifically defined for France requires only a postal code for the post offices box (DEN: *Post Office Box_ Postal Code. Identifier*). The key requirement is that this "*Post Office Box_ Postal Code. Identifier*" is derived from the same BCC that is also used for the postal codes in the German address.

ASBIE – Association Business Information Entity

Since an ABIE is derived from an ACC, the properties of an ABIE also express complex business characteristics. These complex business characteristics may include those already found in another ABIE. The ASBIE represents the association between these two ABIEs. These associations are derived from the appropriate ASCC template. Figure 10 illustrates this concept, with multiple definitions of ASBIEs within ABIEs, all derived from source ASCCs.

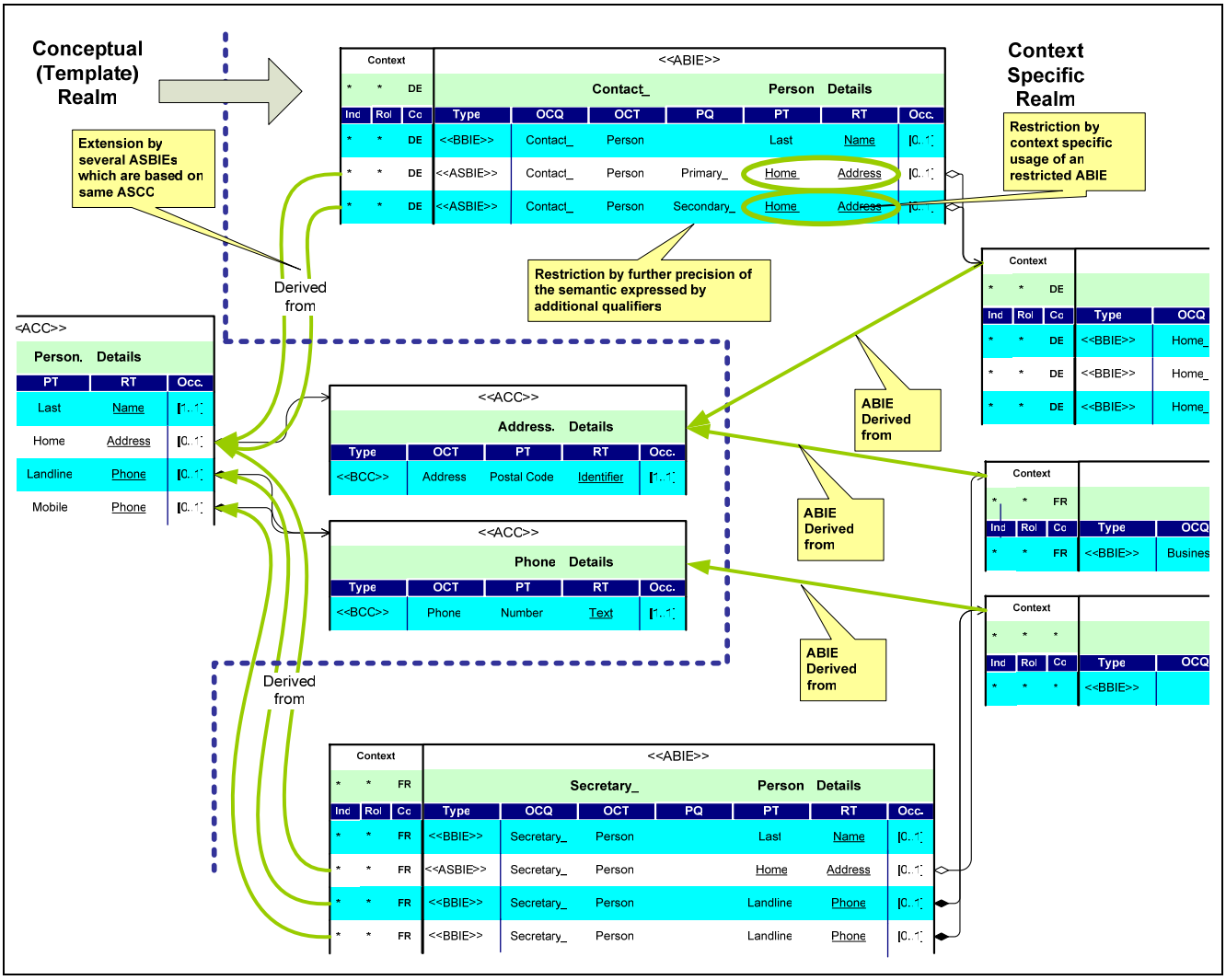


Figure 10 – Derivation and usage of ASBIE

Just as multiple ABIEs can be derived from a single source ACC, and multiple BBIEs can be derived from a single source BCC, multiple ASBIEs can be derived a single source ASCC. For example the "*Contact_ Person. Primary_ Home. Address*" and "*Contact_ Person. Secondary_ Home. Address*" are derived from the same ASCC "*Person. Home. Address*". Additionally, an ASBIE could reflect further qualifications on an existing restricted (qualified) ABIE, like "*Contact_ Person. Primary_ Home. Address*" is based on "*Home_ Address. Details*".

QDT – Qualified Data Types

As we stated previously, a BBIE is always derived from its source BCC. Accordingly, the BBIE must be based on the Core Data Type that is already defined for the source BCC. If the BBIE represents restricted representation of the content or supplementary components, this is accomplished by creating a Qualified Data Type (QDT). Just as ABIEs, BBIEs, and ASBIEs are derived from their source BIE, QDTs are derived from their source CDT. Figure 11 illustrates this concept.

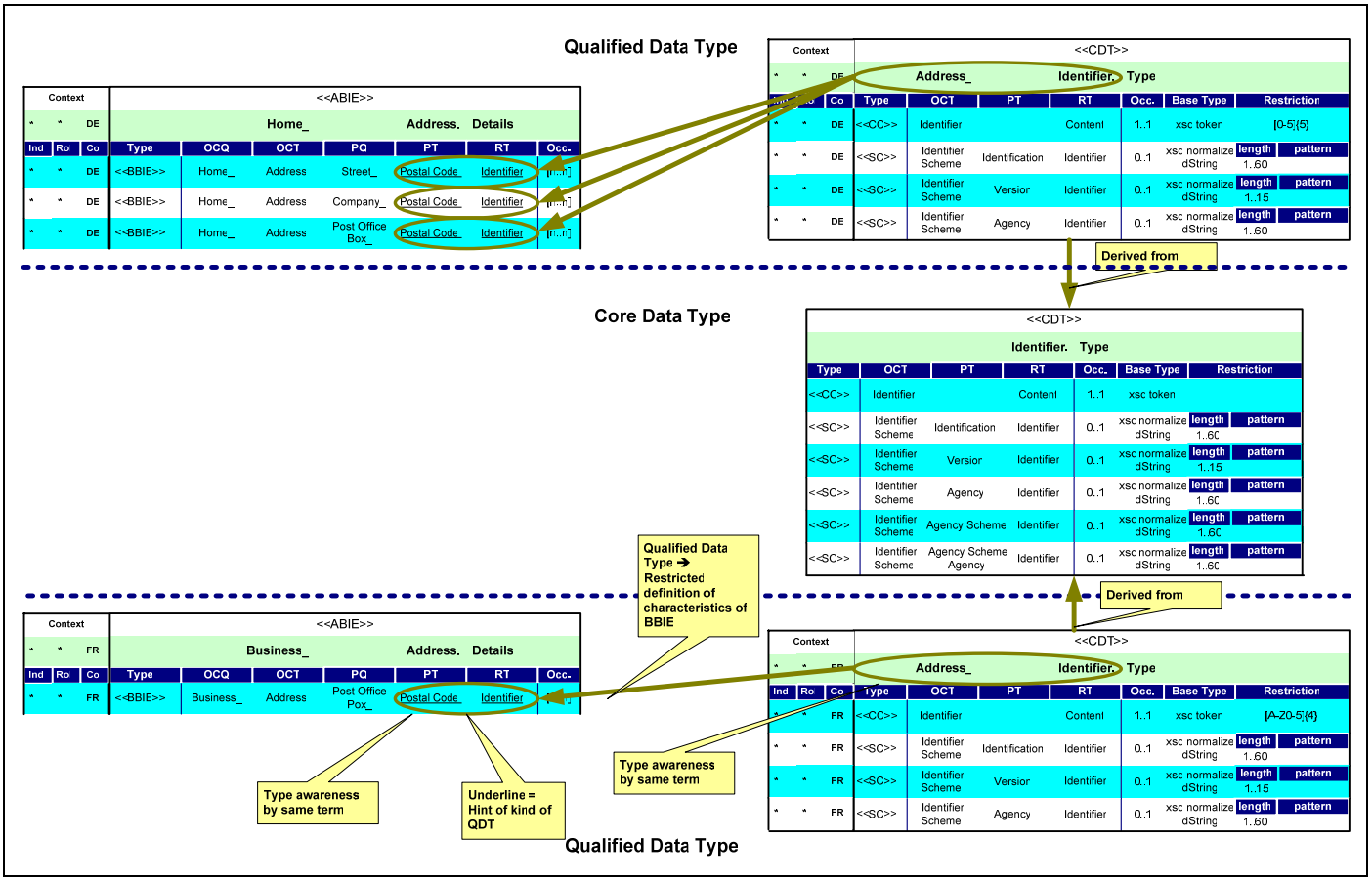


Figure 11 - Derivation and usage of Qualified Data Types

In figure 1, , the BBIE "*Home_ Address. Street_ Postal Code. Identifier*" in the German context is based on the Core Data Type "*Identifier. Type*", as this Core Data Type comes from the BCC "*Address. Postal Code. Identifier*". But if a specific content pattern of German postal code identifier restricted into 5 numbers is required, a Qualified Data Type must be created. The derived Qualified Data Type "*Postal Code_ Identifier. Type*" in the German context is defined to reflect the restricted representation of the content itself. The "*Postal Code_ Identifier*" in the French context could be restricted in the exactly same way.

It is also possible that the Supplementary Components are restricted. As shown in figure 11, the derived "*Postal Code_ Identifier. Type*" contains only 3 Supplementary Components instead of the 5 Supplementary Components from the original "*Identifier. Type*".

Summary

This article has given us a detailed overview of the key model concept in CCTS. In comparison to other modeling languages, CCTS uses a derivation by restriction methodology. This methodology enables the development of similar yet context specific physical/logical Business Information Entities using the conceptual Core Component templates. CCTS aligns to the OO-approach as expressed by UML. As such, the key model concepts of CCTS are easier to understand. However, these key model concepts do not in and of themselves guarantee unambiguous semantically meaningful, context specific business data models. Fixed core data types, consistent naming concepts – such as those identified in ISO 11179, specific modeling

methodologies and detailed context driver principles are also required. We will describe each of these in more detail in future articles to enable you to get a deeper understanding of CCTS and related standards.

The next article in our series will dissect the ISO 11179-based CCTS naming convention. That article will explain the naming conventions in more detail and show how we can get consistent and unambiguous names – regardless of the concept, business semantics, or specific context of the data.

Author Bio

Since his master's degree (MSC, 1993) Gunther Stuhec has worked with communications and EDI technologies. As a consultant in a software house for middleware and EDI systems he developed strategic concepts for customers and was responsible for various EDI projects. He joined SAP SI as a consultant in 1999, where he was responsible for implementing XML/EDI projects in conjunction with SAP systems. Since 2001 Mr. Stuhec works for SAP AG as a “Standards Architect” and has been involved in standardizing business standards on semantical and syntax level.

He is chair of the UN/CEFACT Techniques and Methodologies Group (TMG) that is responsible for the development and maintenance of the UN/CEFACT CCTS standard. He is also a member of various international and national standardization bodies like UN/CEFACT, ISO, and DIN. He is actively involved in developing standards and serves as an interface between these bodies and SAP, introducing SAP's requirements into their work and incorporating their latest findings into SAP's development activities.