TYPE Original Research
PUBLISHED 27 January 2023
DOI 10.3389/fcomp.2022.1008296

Conceptual data systems architecture principles for information systems

Tomas Jonsson*

R&D Department, Genicore, Gothenburg, Sweden

Check for updates

OPEN ACCESS

EDITED BY
Bernhard Thalheim,
University of Kiel, Germany

REVIEWED BY
Maha Khemaja,
University of Sousse, Tunisia
Evellin Cardoso,
Universidade Federal de Goiás, Brazil

*CORRESPONDENCE
Tomas Jonsson
tomas@genicore.se

SPECIALTY SECTION
This article was submitted to Software,
a section of the journal
Frontiers in Computer Science

RECEIVED 31 July 2022
ACCEPTED 08 November 2022
PUBLISHED 27 January 2023

CITATION
Jonsson T (2023) Conceptual data systems architecture principles for information systems.
Front. Comput. Sci. 4:1008296.
doi: 10.3389/fcomp.2022.1008296

Information systems alignment with enterprise evolution affects the performance of enterprises. The systems conceptual and data quality, development time and sustainable life cycle management, are issues for enterprise competitiveness. The ability to directly generate enterprise information systems from models has been thought of as a solution to improve on these issues. Model-driven systems have been a research topic for decades. Fundamental principles for a proven model-driven information systems architecture are outlined in this article. Architectural foundation is a separation of user communities conceptual domain from the information technical domain. The users domain is modeled as an information system model in three layers, conceptual data logic model, interface model and user community model. The technical domain is a platform, allowing the modeling and execution of such a model. These principles have been applied in practice and proven viable. Two platforms and applications, which adhere to these principles, are briefly described.

KEYWORDS

information system architecture, model language, model driven system, conceptual model, model transformation, model execution, model suite

1. Introduction

Enterprise Information Systems (EIS) are sociotechnical systems where the Information Technology (IT)-based Information Systems (ITbIS) and actors communicate with each other. Enterprises are organic entities, which continuously undergo changes due to internal and external factors, Rouse (2005). To maintain the alignment between ITbIS and the enterprise, reconfiguration of ITbIS has to be quick and precise. Misalignment impacts the enterprise performance, Ullah and Lai (2013) and drives the need to replace systems with associated costs and risks. A second replacement driver is the reliance on technical platforms, which go out of date.

Another aspect of ITbIS is the user interaction, Wegner (1997) and Wegner and Goldin (2003), providing users with an interactive information service, Goldin et al. (2000) which supports and empowers them. Such a service should provide information, which is meaningful and correct. Meaningful information contains structures of concepts and data, which are aligned with the users conceptual space, coupled to the natural language of users.

In this article, we describe the general principles for information systems development and the Conceptual Data Systems Architecture (CDSA). The CDSA provides meaningful structures of data to users, with increased conceptual precision, and at the same time simplifies the development and Life Cycle Management (LCM) of ITbIS. CDSA is a layered systems modeling and execution architecture focusing on data-related enterprise concepts and user interaction.

The development of CDSA was influenced by the school of Scandinavian informatics, which was rooted in the works of Langefors (1980). It was also influenced by the Scandinavian school of object orientation, in particular, the idea of a system object model with subsystems as perspectives of the model, for different actor roles within an enterprise, Reenskaug (1977). The definition of the model logic and execution was influenced by functional programming, in particular, the lack of side effects and the notion of lazy evaluation.

The fusion of these ideas led to an architecture with data objects as the metastructure for conceptual information modeling. A conceptual model which will contain all relevant enterprise data concepts and related logic, for a certain ITbIS. Additionally, there is a facility for modeling perspectives of the conceptual model, to satisfy conceptual spaces of various user roles and activities.

Tools development for modeling and code generation were influenced by the Mjølner Project, in particular their metasystem environment, based on a particular kind of object-oriented abstract syntax tree, Madsen and Nørgaard (1987). This object-oriented metaprogramming system was used to create the first version of a conceptual data object modeling language in 1997.

2. From users conceptual domain to IT domain

Enterprise Information Systems are sociotechnical systems where ITbIS and actors communicate with each other. Communication is about exchanging meaningful messages of data. The value of an ITbIS, from a user perspective, is to a high degree dependent on the quality of the conceptual structure and the data it communicates. For ITbIS developers however, focus is on the computational processes that deliver the desired conceptual structure and data.

Developers and users accordingly have different perspectives and conceptual spaces of computer systems (Figure 1). One of the most common reasons for unsuccessful ITbIS development projects is the inability of developers to capture the conceptual space of users and produce adequate systems requirements, Sumner (1999) and McManus and Wood-Harper (2007). This leads to end results which are not satisfactory, in terms of function, time and cost.

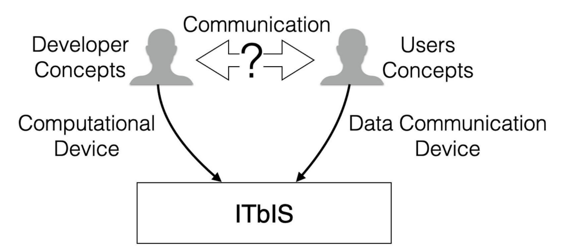


FIGURE 1
Two perspectives and conceptual spaces of information systems lead to communication problems.

Conceptual models play an important role in the development of information systems, Embley and Thalheim (2011) and Thalheim (2012). These models are used to capture and reflect relevant conceptual structures for a user community in an enterprise. Conceptual models play an important role in the communication between users, domain experts, modelers and system designers, in the LCM of ITbIS. Thus, the conceptual model has an important role in the requirement process. However, the path from conceptual model to executing IT system can be a long manual process, requiring design and programming craftsmanship.

The transformation process, from user conceptual domain to target ITbIS, in traditional approaches (Figure 2), involves several steps of manual transformations. Initially, a conceptual model could be made which together with functional requirements becomes the system requirements. From this, a system design is made, including data base design, functional logic design and user interface design. The system design is then coded in one or several languages, depending on the underlying technical environment. The technical environment is made up of compiler, database engine and other frameworks.

Each step involves several degrees of freedom for interpretation and implementation compromises. This leads to deviations from the conceptual space of users.

Time from an established conceptual model to a deployed system is often in the order of years. Since enterprises change continuously, a deployed system is in some respect, misaligned already from deployment. Alternatively, development is troubled with creeping requirements when changes in the enterprise are forced into the development cycle, delaying deployment and increasing cost.

Changes to ITbIS is driven from two different sources, changes in users conceptual domain and changes in the target environment. Enterprises are organic in the sense that they continuously evolve and change. For ITbIS to be aligned with the enterprise, LCM needs to be continuous and synchronized between the enterprise and ITbIS. In target technical environment, changes of various components, such as operating systems, database engines and other components,

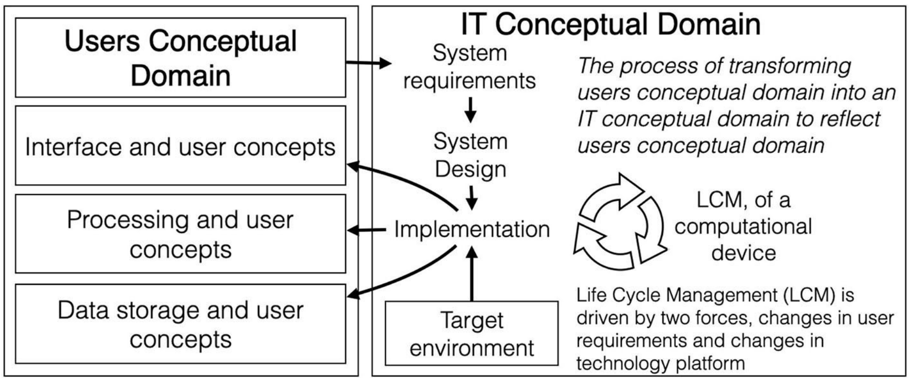

Users Conceptual Domain
Interface and user concepts
Processing and user concepts
Data storage and user concepts
IT Conceptual Domain
System requirements
System Design
Implementation
Target environment
The process of transforming users conceptual domain into an IT conceptual domain to reflect users conceptual domain
LCM, of a computational device
Life Cycle Management (LCM) is driven by two forces, changes in user requirements and changes in technology platform

FIGURE 2
Traditional architecture of conceptual domains for information systems and its life cycle management.

require major rework or reimplementation of the system. This requires resources, takes time and is a cause for alignment disruption.

When systems are coded in some programming languages, the code contains a weave of user concepts and technical concepts such as database, interface and framework concepts. The LCM of such implementations thus becomes a very complex issue.

3. Principles for model-driven information systems architectures

3.1. Information, data and models

In this context, it is useful to provide a definition for *information* and its relationship to models, conceptual models in particular.

Langefors stated the infological (Equation 1) (Langefors, 1980). The message of the equation is that there is a correlation between I information conveyed, S the receiving structure, i.e., the conceptual structure of the mind of the user and D data, i.e., communicated structure from ITbIS, as would be defined in a conceptual model.

For information to be conveyed, data communicated with the users should match the S structure of users pre-knowledge. Since S is users knowledge, conceptual modeling for ITbIS can be considered as a form of knowledge modeling.

$$I = i(D, S, t) \tag{1}$$

D = data represent the intended information
S = the “receiving structure” (pre-knowledge) of the user
t = the time available for the user to interpret data D
I = information conveyed by the data D
i = the information function

3.2. Separation of conceptual domains

In Denning (2003), Denning discuss the principles of computing and separation of the general principles from technologies. In Jaakkola and Thalheim (2011), the authors discuss general principles for WEB information systems, in the dimensions of model and execution architecture. In both cases, a separation of technical and conceptual domains is argued to be independent from each other in respect to the development and LCM (Figure 3). The glue between the domains is the modeling framework.

With a model-driven IT system, i.e., model execution, the development process and LCM will be free from manual transformation processes. Development will be simplified and consistency between conceptual model and implementation can be guaranteed, Jonsson and Enquist (2017). Changes in the users domain, which is continuous, can now be managed as reconfigurations of the system model, without a need for technical reprogramming. This allows for maintaining the ITbIS over time, corresponding to the life cycle of an enterprise.

The strive for domain separation has resulted in several semantic modeling languages with the ambition to generate ITbIS from conceptual models such as ADAPLEX, TAXIS and GALILEO, Borgida (1985). Some architectures have also been developed based on this separation, such as Model-Driven Architecture (MDA) from the Object Management Group (OMG, 2014).

In a literature review of the experiences from applying the model-driven engineering and model-driven development in the industry, Mohagheghi and Dehlen (2008), there where few cases of significant productivity and software quality increases. However, the positive exceptions were cases based on domain-driven development and domain-oriented languages from the telecom industry. Another domain-oriented successful case is that of Carmen Rave Modeling Language, for Crew Roster modeling and system generation, Kohl and Karish (2004).

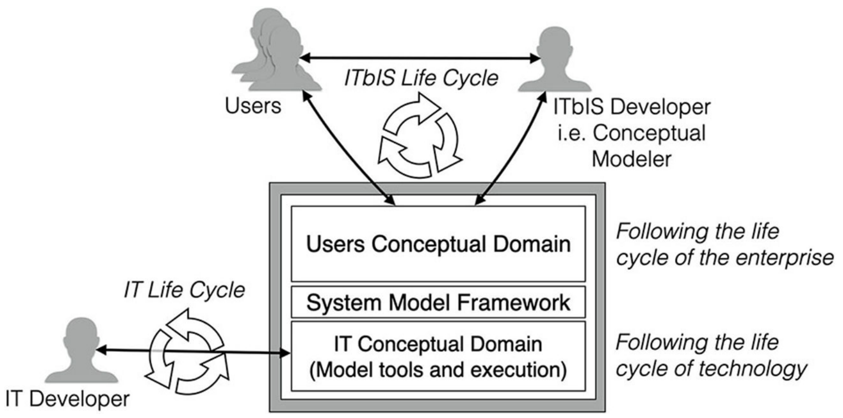

Users
ITbIS Life Cycle
ITbIS Developer i.e. Conceptual Modeler
Users Conceptual Domain
Following the life cycle of the enterprise
IT Life Cycle
System Model Framework
IT Conceptual Domain (Model tools and execution)
Following the life cycle of technology
IT Developer

FIGURE 3
Architecture with separation of conceptual domains for information systems, allowing for independent life cycle management in these domains.

We firmly believe that domain-driven development in conjunction with code generation is a way forward in software development. Our proposed architecture CDSA is a domain modeling and execution architecture, where the domain is *ITbIS*.

The OMG MDA presents some general principles for an architecture of three model layers. Computational Independent Model (CIM) represents the *enterprise domain objects*. Platform-Independent Model (PIM), a technology-independent computational information systems model, and Platform-Specific Model (PSM), a platform technology-specific information systems model. The general idea is to start with a CIM and then transform it to PIM and then to PSM and finally to executing system code. The ambition is to automate these transformations, in particular from PIM to PSM to code. This basic idea seems to be a structured and attractive proposal.

However, OMG also proposed and standardized using Unified Modeling Language (UML) as languages for these models. UML is a rather complex package of 14 types of diagrams plus logic language, describing various aspects of a system. It then becomes a very complex task for tool vendors, based on these various aspects of a system, to verify consistency and then to generate a coherent system. As of September 2022, OMG states, OMG (2022) that typically only 50–70% of PSM can be generated from PIM.

Our approach to system modeling is somewhat different from the mentioned approaches and yields 100% transformation from model to executing system. We propose a model suite with three layers. A conceptual model of *enterprise domain objects* provides for a combination of conceptual structure and declarative functional logic. We add to this, one layer of interface model and one layer of actor role model.

To put it in MDA terminology, we model the CIM as an enterprise object model with both structure of concepts and logic constraints (not computation). Then, we add interface and actor models to define how and by whom concepts in the CIM should be communicated, this is now the PIM, still computation free. This PIM can be automatically transformed into executing system code by a technical platform. So, only one transformation is needed, instead of the three in MDA.

4. Principles for CDSA system model architecture

ITbIS, in the context of enterprises, exist in an environment of different user roles. We consider these systems from four perspectives, data concepts, logic, data users and data interface. To describe an ITbIS, CDSA defines three interconnected layers of models with different modeling language principles (Figure 4).

4.1. Structure of users data concepts domain

The foundation for the architecture is the conceptual model of users data domain, i.e., enterprise objects, which is a unified structure of data-related concepts of all intended users. In its basic form, the model language is process free, as the role of the information system is to provide information to actors in the enterprise, while actors drive the enterprise processes. In some cases, a state diagram model component could be added to express the object life cycle, as a means for explicit sequential control.

4.2. Users with conceptual domains

Users are part of the sociotechnical EIS and communicate with ITbIS. For ITbIS in an enterprise context, there will be users

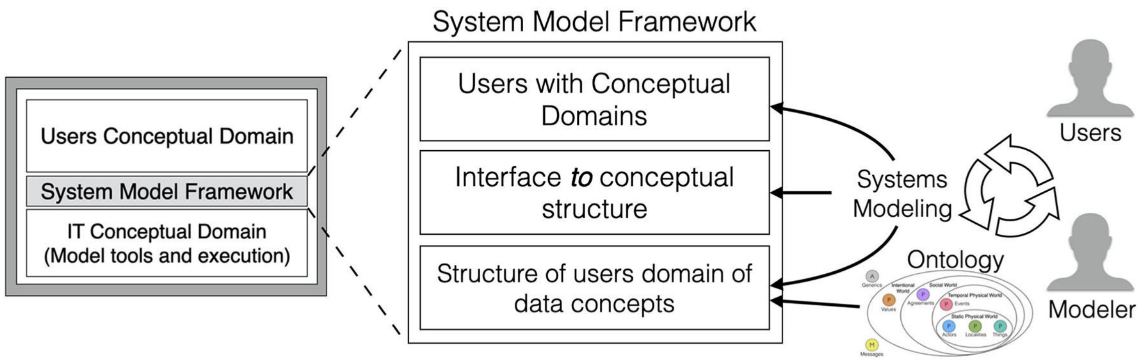


FIGURE 4
System Model Framework Architecture in Conceptual Data Systems Architecture (CDSA), with its three-layer model suite.

with different roles and interest of concepts and data. Different roles in enterprises affect usage of the ITbIS, so a model of user roles is needed.

4.3. Interface to conceptual structure

To describe the communication between the technical system and the social system, messages need to be defined, adhering to the conceptual structure of users, i.e., the conceptual model. We refer to these messages as views, which in practice can have different forms, such as a layout for computer screen, document, or XML format definition. As different roles of an enterprise focus on different concepts and data, certain sets of messages should be related to certain roles.

4.4. Principles for CDSA conceptual models

4.4.1. General principles

The language for CDSA conceptual models is a Conceptual Data Logic Language (CDLL). CDLL can be described as a domain model language, where the domain is information. CDLL is object oriented, including object types, inheritance, polymorphism, data properties, relations and logic. An essential difference from traditional object-oriented languages is that there are no processing concepts such as methods or functions related to the object type. Instead, logic is added to attributes to define their value and state. Furthermore, logic can only affect state and value of the attribute it is defined on, thus the language is pure functional in respect to lack of side effects.

As can be seen in Figure 5, the CDLL meta model is divided into three basic components, *concept declaration*, *attribute definition* and *concept reference*.

4.4.2. Concept declarations

By using declarations of object types with attributes, it is possible to declare a structure of concepts based on the notion of objects. Additionally, there can be concepts other than object types and attributes, such as, for example, states, called static concepts. All concepts can be referenced and included in logic statements explained below.

4.4.3. Attribute definition

Attributes are in this kind of language more than a slot for data, with both additional states and logic expressions.

4.4.3.1. Type and dimension

Dimension applies to attributes and typically there will be 0-, 1- and 2-dimensional attributes, although there could be higher dimensions. Dimension 0 represents a single value, dimension 1 is a set of values and dimension 2 is a set of sets of values.

Relations are binary, relating to another or the same object type. Relation attributes have the corresponding object type with a dimension of 0 or 1.

Property types can be considered from a conceptual or a computer science perspective. From a conceptual perspective, a property can be a measurement, which can be represented by a real number datatype in a computer. Following is a list of data types that are typically part of a CDSA platform.

- Measurement - Real number and unit label
- Countable - Integer
- State - Boolean
- Text - String
- Timepoint - Integer (seconds related to a certain start time)
- Color - Integer x 3 (red, green and blue)
- Media - String (file path or URI) or media itself

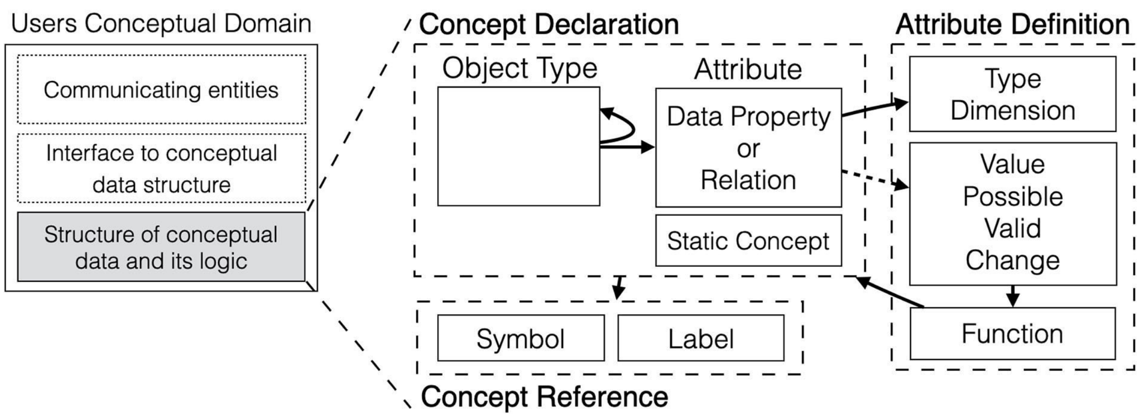


FIGURE 5
Conceptual Data Logic Language (CDLL) meta model overview with its three main components, Concept declaration, Concept reference, and Attribute definition.

In addition, a property can have the type of an object-type reference.

4.4.3.2. Logic

In Figure 5, it is indicated that there are four slots for adding logic to an attribute, value, possible values, valid value and changeable state. In certain implementations, additional slots have been applied, such as readable, can-add value, can-delete value, etc.

Logic is defined by some language, either using the language of the target platform such as java or C++ or using a logic language designed for the architecture. In case of target platform language, certain restrictions apply, specifically the logic should form an expression or algorithm returning a value, without causing side effects. The advantage of using a target environment language is that it is easier to implement code generation from the model, however a logic language properly designed for this type of model would be more appropriate.

4.4.4. Example model

In Figure 6, an object-type diagram of a model is shown, in the notation style, i.e., graphical syntax, we developed. This diagram does not display properties and is intended for model navigation in large models, in some modeling tool. Sub-types are located inside the super-type as subsets of a set. That is, an object type represents a set of instances and a super-type represents the union of all sub-type sets. A consequence is that this, by default, cluster type hierarchies together, for easier navigation in large models. Property declaration and attribute definitions would be handled in separate panels or windows, by first selecting the object type of interest.

Relation attributes are either single value (dimension 0) with a square symbol or set (dimension 1) with a circle symbol. UML

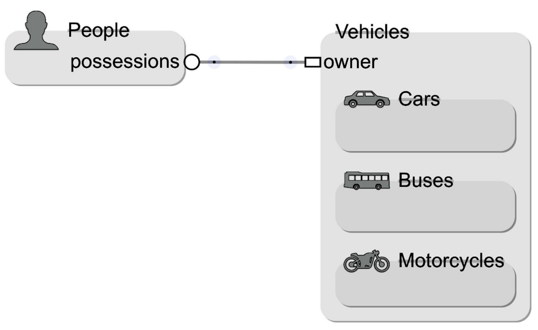


FIGURE 6
Object-type diagram of an example model. Notation shows object types with label, icon and relation attributes. Sub-types are constrained within the boundaries of their super-type. Other notations could be used. Complete model is listed in Listing 1.

class diagram style including properties could also be used if desired, as the language itself does not impose specific graphical notation, it is the meta model principle that is of importance.

In Listing 1, the complete model is shown, with properties and some function definitions. For *Vehicle* type, a *possible* and a *change* function is added and for *Person* type *possessions value* function is added.

```
system ’People - Vehicle’ language ’en’
baseModel
  ’Vehicle’ thing
    ’brand’ text
      possible ["SAAB", "Volvo", "Mercedes",
              "BMW", "Kawasaki"];
    ’model’ text
    ’registration number’ text
```

```
      change system.’current user’.role.
          name = "registration admin"
    ’value’ number "€"
    ’picture’ file
    ’owner’ ’Person’ as ’possession’
  end ’Vehicle’

  ’Car’ thing kindof ’Vehicle’
    ’Power’ number "hp"
  end ’Car’

  ’Bus’ thing kindof ’Vehicle’
    ’number of passenger’ integer
  end ’Bus’

  ’Motorcycle’ thing kindof ’Vehicle’
  end ’Motorcycle’

  ’Person’ person
    ’given name’ text
    ’family name’ text
    ’possession value’ number "€"
       value sum(possession.value)
    ’possession’ ’Vehicle’ set as ’owner’
  end ’Person’
end baseModel
end system
```

Listing 1 CDLL example listing as generated from a modeling tool.

4.4.5. Concept reference, symbols and labels

With labels, we mean a word or a natural language expression that is used to refer to the concept. For multilanguage user environments it is preferable to have support for defining labels using different natural languages, for the same concept. Symbols can also be used for concept reference and has been used mostly for object types as alternative type identification in modeling tools and screen-based interfaces.

4.4.5.1. Natural language and model of concepts

A labeling method has been developed, to couple concept declaration structure to semantics of natural language. Rules for constructing English language sentences from the model have been developed and some of them are described below. They are used to validate a consistently labeled model and to generate a dictionary of declared concepts, with generated statements about the concept. These statements can be used in an LCM process to test users acceptance of concept definitions and labeling.

- *has rule* - <object type>has <attribute>[which can be <possible>]
- *can be rule* - <object type>can be <state>
- *kind of rule* - <object type>is a kind of <super type>
- *which can be rule* - <object type>which can be <sub types>

4.4.5.2. Example dictionary

```
Dictionary for People - Vehicle

brand [n,pr] vehicles have a brand which can
be for example SAAB, Volvo or Mercedes
bus [n,ot-thing] buses are some kind
of vehicles
car [n,ot-thing] cars are some kind
of vehicles
family name [n,pr] people have a family name
given name [n,pr] people have a given name
model [n,pr] vehicles have a model
motorcycle [n,ot-thing] motorcycles are some
kind of vehicles
number of passenger [n,pr] buses have number of
passengers
owner [n,r] vehicles can have an owner which
is a person
person [n,ot-actor]
picture [n,pr] vehicles have a picture
possession [n,r] people can have many
possessions which are vehicles
possession value [n,pr] people have possession
value
power [n,pr] cars have power
registration number [n,pr] vehicles have a
registration number
value [n,pr] vehicles have value
vehicle [n,ot-thing] A vehicle is either a
car, a bus or a motorcycle.

Format description

word [-,-]
[n,-] noun, [adj,-] adjective
[-,ot-*] object type-object kind, [-,r]
 relation, [-,pr] property
```

Listing 2 Example of generated dictionary.

4.4.6. Execution principle

The execution environment shall allow for object instance creation and ensure that all instances of objects are persistent and searchable, until explicitly deleted. Changes in object states shall be distributed to all parties with an interest in such state and a mechanism for transactional integrity needs to be in place.

Since the CDLL model is strictly functional from the perspective of attributes, principles of data-flow execution can be applied, Johnston et al. (2004) which could give an additional benefit of intrinsic parallelism. Data dependency graphs are generated from the logic expressions. Execution engine or code generation handles the data-flow as appropriate (Figure 7).

4.4.7. Conceptual modeling guidance

The architecture has both a conceptual model layer and a (conceptual model) view layer. It is important to take this into account when creating the conceptual model. Meaning, modeling the conceptual objects in themselves, and not perspectives of these objects. For instance, at a university there are students, teachers, researchers, administrators, etc. Modeling them as separate type entities would in this case be wrong, since they are all people who can appear in one or more roles. Information perspectives of people in different roles should be represented with different views of person information.

The modeling language and its application in the modeling process are in our case guided by the Phenomenological Foundational Ontology, briefly described in Jonsson and Enquist (2019). The ontology is founded in the phenomenological system of Edmund Husserl (1859–1938), as described in Woodruff Smith (2003). The phenomenological system reasons about what exists, mental objects and related ontological regions. The ontology guides modelers by giving them possible categories of neutral (nonperspective biased) object types, such as thing, actor, location, event, agreement and value.

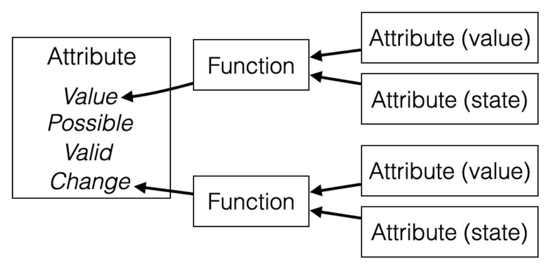


FIGURE 7
CDSL data-flow execution principle where reevaluation of a function occurs as appropriate, either when an input value is changed and/or when the result is accessed. Arrows indicate the direction of data-flow.

4.5. Perspectives and interface principles for CDSA systems

An enterprise consist of actors working together creating something of value. Different actors, or groups of actors, have different conceptual perspectives of the enterprise. To collaborate they need to communicate, i.e., there needs to be some coupling between users conceptual spaces. Here, it is in place to point out that not all enterprise actors conceptual spaces, necessarily should be included in one system. Which

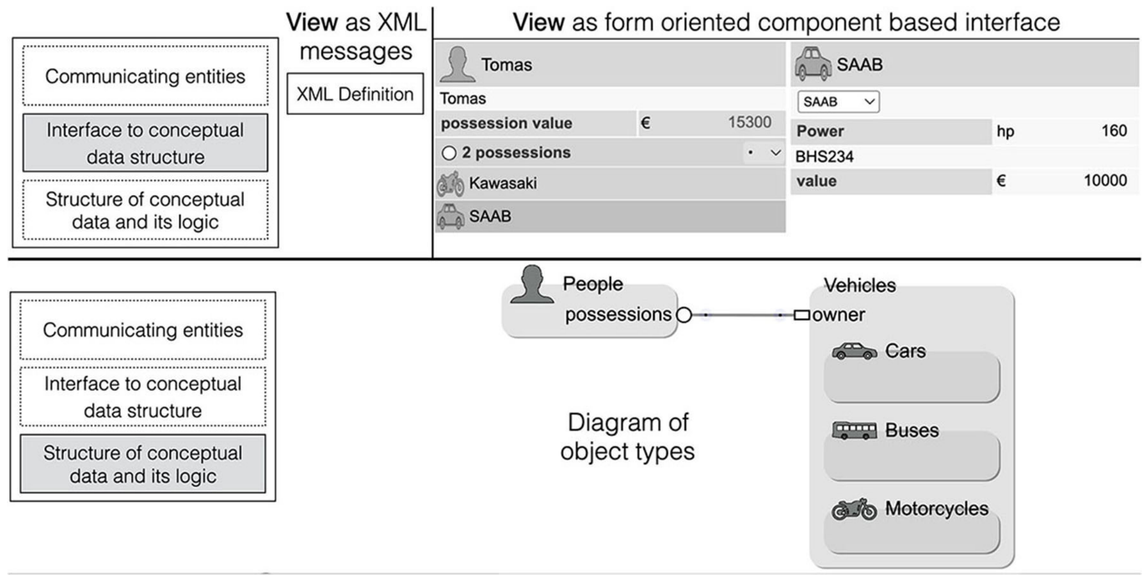


FIGURE 8
Views principle illustrated as interfaces to a corresponding conceptual model. The components of a form-based interface model language are object views, relation views, and property views. Each view component is related to a component in the object-type model. During system execution, interface panels are populated with data from object instances, as selected by the user.

actor perspectives to include in one system depends on how conceptually coupled they are and how frequent communication is between them.

If the ITbIS system is to be compatible with different enterprise perspectives, the system modeling facility needs to include a perspective-modeling component. We call these perspectives for views. It is through views that users communicate with an ITbIS. A view can thus be regarded as a message format definition, which is comprised of semantics and syntax. The semantics is the conceptual content and the syntax is the format. Views are mostly formed as a hierarchy of object components, always adhering to the semantics of the conceptual model, starting from a root object.

In Figure 8, a simple form-oriented syntax is shown. For each concept component type of the conceptual meta model, there needs to be at least one corresponding interface component type, representing a syntactic structure. A view is then a composition model of interface components, relating to components of the conceptual model.

The same principle was advocated in the Naked Object approach, Pawson (2004). In this case, one view per object type is autogenerated from an object-type structure. Similarly in CoreWEB, Jonsson and Enquist (2017) a default view is generated with a view panel for each object type. Additionally, custom views can be configured and represented in a views model.

For such interactive screen-oriented user interfaces, there could be different syntaxes, i.e., styles. From simplistic form styles as shown, to more complex form styles with several alternative interface components for each meta model concept component, including various kinds of graphics. However, the semantic structure of the conceptual model should be preserved and be apparent in the interface. For instance, a property concept should always be conveyed in relation to the object to which it belongs.

To provide a machine-to-machine interface, XML-based views could be used with default or configured views of the conceptual model. When communicating with other conceptual domains, such as external enterprises, conversion of conceptual spaces would be required.

4.6. User model principles for CDSA

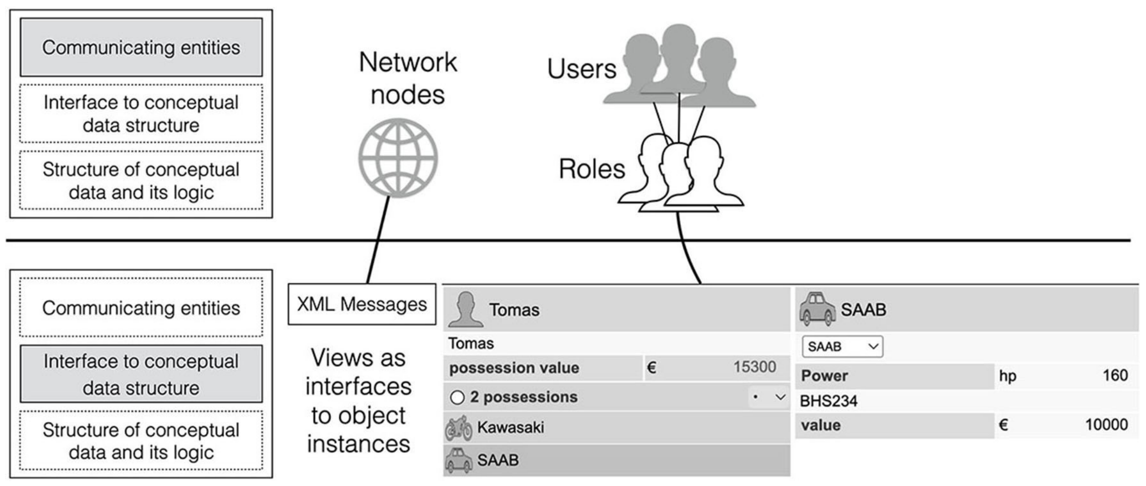
There are two primary types of users of ITbIS, people and other IT systems (Figure 9). For people in the enterprise context, we define roles, which are related to specific views and then assigned to individual users. Similarly for external IT systems, a set of message formats are defined and assigned for communication with a specific network address. This is the basic principle of a user model, sufficient in many cases.

Beyond basic principles and the scope of this article, there can be both role structures, organizational structures and activity models, as represented by enterprise architecture models.

5. Results

5.1. Summary of principles

Following is a summary of the three basic principles for CDSA.



Communicating entities
Interface to conceptual data structure
Structure of conceptual data and its logic
Network nodes
Users
Roles
Communicating entities
Interface to conceptual data structure
Structure of conceptual data and its logic
XML Messages
Views as interfaces to object instances
Tomas
Tomas
possession value
€
15300
2 possessions
Kawasaki
SAAB
SAAB
SAAB
Power
hp
160
BHS234
value
€
10000

FIGURE 9
Principle of user model connecting the social world to the interface of the Information Technology based Information Systems (ITbIS), completing a model of a sociotechnical system.

- *Separation of user conceptual and technical conceptual domains*, with a user concept system model framework, which is executable by the technical domain.
- *Model suite for a system model framework* in three coupled layers, concept, view and user layer. Concept layer for declaring and defining concepts, view layer for perspectives of conceptual structure and user layer for directing sets of views to users or groups of users.
- *Conceptual data logic language*, declaring and defining data concepts without involving processing concepts. This minimizes the number of model concepts, avoids computational side effects and allows for data-flow execution.

5.2. Platforms and system cases

5.2.1. Core enterprise architecture framework

CoreEAF is a distributed parallel processing CDSA environment for ITbIS, where processing take place on server and clients. The server distributes and coordinates data exchanges between clients. Target languages for the core functionality are C++ and SQL, with additional XML based languages for systems to systems communication and document generation. Some of the capabilities of the related execution engine are described at: http://www.genicore.se/index.htm?page=corepro.

5.2.2. System cases with CoreEAF

CoreEAF was initially developed to support building a customized project planning and management system for the Swedish Defense Material Agency. This system gradually replaced several other systems and grew to a complete Enterprise Resource Planning (ERP) system, Jonsson and Enquist (2015) with over 5,000 user concepts and 3,000 users. The system and platform have been developed and supported over 25 years. The system model updates every 4 months, driven by enterprise evolution and platform updates every 5 years, driven by changes in technical infrastructure. The platform has subsequently been used for a number of other systems in the public sector in Sweden. The platform is also used as a meta modeling facility, to generate new versions of the platform itself.

5.2.3. CoreWEB

CoreWEB is a non-commercial software as a service implementation based on the CDSA, for education and prototyping. CoreWEB allows a conceptual modeler to generate versions of a system from a CDLL model, complete with user interface. The system can further be refined through modeling customized views and user roles. CoreWEB has been used for prototyping, deployed systems and in education at Gothenburg, Rostock (Lambusch et al., 2020) and Kiel universities, as well as demonstrated at some conferences (Jonsson and Enquist, 2017, 2018; Jonsson and Rimfors, 2020). CoreWEB is available at https://ameis.se/cml/index.htm. Some course materials for students are also available.

6. Discussion and future directions

This article outlines the *principles* for a model-based architecture, which could be extended in different directions. The three-layer modeling suite is a domain model architecture, where the domain is IT-based enterprise information systems (ITbIS). It does not address many enterprise architecture issues, such as work processes, goals and strategies. But perhaps, CDSA can become a component in some more general enterprise architecture frameworks.

6.1. Consequences of applying CDSA principles

- *Pure conceptual space.* Development and LCM processes take place in the conceptual space of users, which means that both developers and users focus on the same concepts and talk the same language.
- *Co-creative development process.* By repetitively generating instances of a system from the model during development process, users and other enterprise actors can actively participate in an exploratory and co-creative process.
- *Alignment between the ITbIS and enterprise, maintained over time.* With change cycles from minutes to days in most cases, keeping up with enterprise evolution is not a problem. However, it is important that the user community is mentally aligned with system changes. In our experience, it is recommended to collect and implement modification requests continuously for a verification purpose and to have releases in cycles of 3–12 months.
- *Reduction of complexity.* The functional, non side effect, nature of the model language makes calculation and logic errors trivial to pinpoint. Reduction in the number of statements required for a system also reduces the complexity. The above-mentioned ERP system roughly consists of 13,000 conceptual model statements and 20,000 statements for the form-based interface model, making a total of 33.000 statements. Without any direct comparison case, we estimate that this is in one to two orders (10–100) of statement reduction, compared to traditionally coded systems.
- *Fifth generation programming.* In Thalheim (2021) and Thalheim and Jaakkola (2020) the authors argue that when a program is described and generated from a model, Models as a Program (MaaP), it is a case of 5th generation

programming. For a specific application domain, we have shown how this is possible.

6.2. Future directions

- Work on refining the foundational phenomenological ontology is ongoing.
- Development of a logic language. In mentioned platforms, the target environment language has been used to define logic of the model. In an exploratory project, an expression parallel processing data-flow language and compiler for WEB applications was developed. One future project is to integrate the logic of this language into the CoreWEB environment.
- We aim to expand our community network related to conceptual modeling, modeling to programs and models as programs.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

The author confirms being the sole contributor of this work and has approved it for publication.

Funding

This research was funded by the Genicore AB.

Acknowledgments

All tool and architecture development has been performed in parallel with commercial systems development projects without external funding. In 2015, Genicore decided to disseminate knowledge about the architecture and principles developed, which resulted in the first publication at CAISE, Jonsson and Enquist (2015) in cooperation with Gothenburg University. Later academic engagement led to cooperation with Kiel and Rostock Universities.

Conflict of interest

TJ was employed by Genicore.

Publisher's note

References

Borgida, A. (1985). Features of languages for the development of information systems at the conceptual level. *IEEE Softw.* 2, 63–72. doi: 10.1109/MS.1985.230050

Denning, P. J. (2003). Great principles of computing. *Commun. ACM* 46, 15–20. doi: 10.1145/948383.948400

Embley, D. W., and Thalheim, B. (2011). "Handbook of conceptual modeling: theory, practice, and research challenges," in *Handbook of Conceptual Modeling* (Berlin; Heidelberg: Springer Berlin Heidelberg).

Goldin, D., Srinivasa, S., and Thalheim, B. (2000). "IS=DBS+Interaction: towards principles of information system design," in *Conceptual Modeling—ER 2000, Vol. 1920*, eds G. Goos, J. Hartmanis, J. van Leeuwen, A. H. F. Laender, S. W. Liddle, and V. C. Storey (Berlin; Heidelberg: Springer Berlin Heidelberg), 140–153.

Jaakkola, H., and Thalheim, B. (2011). "Architecture-driven modelling methodologies," in *Information Modelling and Knowledge Bases, Vol. XXII* (Jyväskylä: IOS Press), 97–116.

Johnston, W. M., Hanna, J. R. P., and Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Comput. Surveys* 36, 1–34. doi: 10.1145/1013208.1013209

Jonsson, T., and Enquist, H. (2015). "CoreEAF-a model driven approach to information systems," in *CEUR Workshop Proceedings, Vol. 1367* (Stockholm), 137–144.

Jonsson, T., and Enquist, H. (2017). "Semantic consistency in enterprise models-through seamless modelling and execution support," in *Proceedings of the {ER} Forum 2017 and the {ER} 2017 Demo Track*, volume 1979 of *CEUR Workshop Proceedings* (Valencia: CEUR-WS.org), 343–346.

Jonsson, T., and Enquist, H. (2018). "Phenomenological ontology guided conceptual modeling for enterprise information systems," in *Advances in Conceptual Modeling, volume 11158 of LNCS* (Xi'an: Springer International Publishing), 31–34.

Jonsson, T., and Enquist, H. (2019). "Phenomenological framework for model enabled enterprise information systems," in *New Trends in Databases and Information Systems, Vol. 1064* (Cham: Springer International Publishing), 176–187.

Jonsson, T., and Rimfors, M. (2020). "CoreWEB-semantic expressions in conceptual models for generation of information systems," in *Modellierung 2020 Short, Workshop and Tools Demo Papers, Vol. 2542* (Vienna: CEUR), 208–212.

Kohl, N., and Karish, S. E. (2004). Airline crew rostering: problem types, modeling, and optimization. *Ann. Operat. Res.* 127, 223–257. doi: 10.1023/B:ANOR.0000019091.54417.ca

Lambusch, F., Enquist, H., and Jonsson, T. (2020). "Creating vividness through executable models: a teaching case for conceptual modelling," in *Forum at Practice of Enterprise Modeling 2020, Vol. 2793* (Latvia: CEUR), 13–23.

Langefors, B. (1980). Infological models and information user views. *Inf. Syst.* 5, 17–32. doi: 10.1016/0306-4379(80)90065-4

Madsen, O. L., and Nørgaard, C. (1987). An object-oriented metaprogramming system. *DAIMI Rep. Ser.* 16, 7592. doi: 10.7146/dpb.v16i236.7592

McManus, J., and Wood-Harper, T. (2007). Understanding the sources of information systemsproject failure. *J. Inst. Manag. Serv.* 51, 38–43. Available online at: https://www.ims-productivity.com/user/custom/journal/2007/Autumn/MSJaut07.pdf

Mohagheghi, P., and Dehlen, V. (2008). "Where is the proof?-a review of experiences from applying MDE in industry," in *Model Driven Architecture –Foundations and Applications, Vol. 5095*, eds I. Schieferdecker and A. Hartman (Berlin; Heidelberg: Springer Berlin Heidelberg), 432–443.

OMG (2014). *Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0.* Technical report, Object Management Group.

OMG (2022). *MDA FAQ, How is MDA being delivered? In what kind of tools*? Technical report, Object Management Group.

Pawson, R. (2004). *Naked Objects* (Ph.D. thesis). University of Dublin, Trinity College, Dublin, Ireland.

Reenskaug, T. (1977). "PROKON/PLAN-a modelling tool for project planning and control," in *IFIP Proceedings* (Toronto, ON), 717–722.

Rouse, W. B. (2005). A theory of enterprise transformation. *Syst. Eng.* 8, 279–295. doi: 10.1002/sys.20035

Sumner, M. (1999). "Critical success factors in enterprise wide information management systems projects," in *Proceedings of the 1999 ACM SIGCPR Conference on Computer Personnel Research-SIGCPR '99* (New Orleans, LA: ACM Press), 297–303.

Thalheim, B. (2012). "The art of conceptual modelling," in *Information Modelling and Knowledge Bases XXIII, volume 237 of Frontiers in Artificial Intelligence and Applications* (Tallin: IOS Press), 149–168.

Thalheim, B. (2021). "From Models_For_Programming to Modelling_To_Program and Towards Models_As_A_Program," in *Modelling to Program, Vol. 1401*, eds A. Dahanayake, O. Pastor, and B. Thalheim (Cham: Springer International Publishing), 3–44.

Thalheim, B., and Jaakkola, H. (2020). "Model-based fifth generation programming," in *Information Modelling and Knowledge Bases XXXI* (Lappeenranta), 381–400.

Ullah, A., and Lai, R. (2013). A systematic review of business and information technology alignment. *ACM Trans. Manag. Inf. Syst.* 4, 1–30. doi: 10.1145/2445560.2445564

Wegner, P. (1997). Why interaction is more powerful than algorithms. *Commun. ACM.* 40, 80–91. doi: 10.1145/253769.253801

Wegner, P., and Goldin, D. (2003). Computation beyond turing machines. *Commun. ACM.* 46, 100–102. doi: 10.1145/641205.641235

Woodruff Smith, D. (2003). "Pure" logic, ontology, and phenomenology. *Revue Internationale de Philosophie* 2003/2, 21–44. doi: 10.3917/rip.224.0021