UML tailored to an ERP framework

Jesper Linvald, IT-University of Copenhagen, linvald@it-c.dk (corresponding author)
Kasper Østerbye, IT-University of Copenhagen, kasper@it-c.dk

Abstract
Enterprise Resource Planning (ERP) systems are typically developed based on a large framework. This framework is customized by partners to provide solutions to customers. It is therefore of interest to make the development process for the partners as efficient as possible and one way to do this is to provide them with specialized modelling tools. This paper presents our ideas for such a domain specific modelling environment, where the graphical notation is inspired by UML, but incorporates specific aspects unique to the specific framework we are considering.

Keywords
ERP, UML, domain specific model and language, graphical configuration, intentional programming, AST, CASE tools.

1 Introduction
We are currently developing a graphical configuration tool for an Enterprise Resource Planning system (ERP), which is implemented as a .NET solution. The existing and working solution captures the essentials of the entire ERP-domain and encapsulates the concepts of a business on an abstract level.

This paper presents a project which has as its goal to investigate if an UML-like tool can be designed to provide the same functionality as the current solution, but in a more efficient manner. The primary issue we address is that of maintaining an overview of a full solution.

We present an overview of the domain model; this presentation will be quite superficial, and is not intended to give the reader a full understanding of the rationale behind the model, but will provide enough information for the reader to appreciate the problems and our solutions. We will then present a proposal for a graphical syntax for the problem domain, a syntax that is inspired by UML class diagrams, but also incorporates the domain specific elements.

2 The domain specific object model
The specific ERP model allows the definition of business object classes. Each class is build by describing a set of elements that corresponds to primitive constructs in the underlying framework. One way to think of this, is that the model allows for the definition of arbitrary classes, but rather than having integer and String be the primitives of the model as in Java, the primitive elements are elements like address, running total including VAT, deadlines etc. These are the cores of the ERP knowledge captured in
basic building blocks called elements. These elements each have a number of predefined attributes that must be configured as part of the development process.

2.1 Containment and relation

The model also allows relationships to be defined between classes. To keep the definitions object centric, relations are also specified using elements. Elements representing the different roles of a binary relation can be tied together. Only one-to-many relationships can be defined. Containment, a special sort of relation, has given name to the two generic roles of a relationship, “master” and “detail”. If the master object in a containment relation is deleted, all the contained (detail) objects are deleted as well. The names “master” and “detail” are also used in general relationships, where the master is the object with single cardinality, and the details are those with the multiple cardinality. The general relationships do not have the same deletion semantics as containment.

One of the obvious choices in a UML inspired representation is to represent relationships in a graphical manner similar to associations, as opposed to the current model where the master and detail elements are configured as any other element type, through filling out property boxes.

![Diagram](image)

Figure 1: A small UML example of the possible types of relationships. Each pair consists of a master and a detail element, e.g. the relationship between SalesOrder and Customer is represented by two elements, Customer of type relationmaster located in SalesOrder, and SalesOrder of type RelationDetail in Customer.

3 Graphical configuration

In the current tool, a business solution is created and configured manually by dragging and dropping business object classes and elements into a tree like structure in Visual Studio .NET. All the predefined classes and elements are available in the toolbox and the properties of the different elements are modified in the property box (Notice that in Visual Studio .NET, there is one such property box opened at any given point in time). The lack of direct graphical representation of the relations and containment, as
well as for the single property box makes it hard to track relationships between objects.

The purpose of this project is to facilitate the design activity by conveying an opportunity for visual UML-like composition and configuration of the predefined classes and elements already existing in the solution and thus facilitating the modelling of the final specialized ERP solution and at the same time lending support to the conceptual model of the entire domain.

3.1 The “halo” metaphor

The problem in directing all configurations to the toolbox is, as mentioned above, that only one element can be explored at the time as well as no general overview is achieved through this approach. At the same time the mental model of the developer, who should conceive that properties as living on the objects, is not supported.

To solve this we have among other things made use of a graphical metaphor known as a “halo” as found in the Java based UML CASE tool “Visual Paradigm” [4] (see figure 3).

![Figure 2: A Visual Paradigm halo (the icons surrounding AClass).](image)

Each entity has a halo of icons attached to it, which becomes visible on the mouse-Over event, which strengthens the confidence in what entity the developer in fact is dealing with. The halo represents the possible and legal configuration options of the entity selected.

These configuration options of the object are actualized simply by dragging them to a different location where a new object is instantiated. For example dragging in the specialization arrow will create a subclass of “AClass” if released over empty canvas, or making an existing class a subclass if released over an existing class. The level of support gained from this metaphor is equivalent to the one offered by intellisense\(^1\) of code editors when coding.

---

\(^1\) Intellisense: the popup of possible methods and fields when i.e. typing a dot in Java.
We find the halo to be advantageous, and will pursue it further in the system. Another convenient aspect of the halo system over the toolbox approach is that it minimizes mouse movement, especially on large screens.

3.2 Our visual domain specific configuration model

The representation of a class in UML is good for representing a named entity serving as a logical container as well as a self contained unit and we take our outset in than metaphor. However, extending the looks of this container as well as the elements contained within it might greatly improve the readability and overview of diagrams as well as strengthen the conceptual model of the domain.

Despite the fact that we prefer halo’s to toolbars, it should be mentioned that two toolbars has been provided for because we recognized the need for a central repository for all the objects, elements and configuration possibilities in addition to having them on the objects themselves. When a business class is selected, those elements which are in not possible are greyed out, giving a repertoire corresponding to those available in the halo. These toolbars are rather ordinary and want receive any further attention.

3.3 A refined UML model

We will now detail our design, referring to the large figure 3 on next page.

The colour scheme:

In large solutions with many objects colours are often more expressive than words, which is why, a colour scheme is attached to each entity\(^2\). The relation lines are drawn in the colour of the master element making it easy to determine the master/detail relationship. The choice of colour nuance stems from the arch type of which there are four (Party, Asset, Line item and Contract) – too many different nuances would work against the ambition of overview.

The title bar:

The title bar depicts the class of the business object. The red colour along with the dots indicates that information has been left out and that the developer may expand any additional information by clicking the red area. The “D” icon at the far left allows for setting up default values of the object.

The button to the far right reading “DOC” represents the documentation for the object. When clicked, a textbox is opened/inserted between the title bar and the attributes part where documentation and comments can be inserted.

Attributes:

This area consists of three entities:

1. A drop down list of all possible attributes/elements, which are allowed to be added to the business object.
2. An accept button which inserts the selected element in expanded state into the object allowing for direct manipulation of the properties.
3. The elements that have already been inserted into the class. These can be expanded as well by double-clicking them. The address element is expanded in this way.

\(^2\) Most UML CASE tools do not make any use of colours or colour schemes which is a shame because it greatly improves readability and overview. Borland’s JBuilder is an example of a CASE tool that in our opinion displays UML diagrams in colours successfully.
Relations:

Relations, both master and detail, have been extended out of the box suggesting their involvement with another part of the model, and also allowing us to show the design without association lines, and still have a clear indication of relations. The colour of the master and detail elements are the same, and dictates the colour of the line, making it easy to follow the dependencies with the eyes.

Right clicking the relation object will present two opportunities:

1. “Isolate the two” giving an opportunity to abstract from the rest of the model by toning down all irrelevant material.
2. “Go to master/detail” which allows for easy navigation between the involved parties. If clicked the canvas scrolls to and focuses on the other end of the relation.

![Diagram](image_url)

Figure 3: The containment object expanded inside the parent object. The relation to and the Customer object is shown as well. The figure does to reproduce nicely on black and white printers. See the colour figure at [www.it-c.dk/~linvald/programming/UML.html](http://www.it-c.dk/~linvald/programming/UML.html)
A small “M” or “D” signifies if the element represents a master or detail. These symbols can be modified directly in place. Icons displayed on each relation element assist in identifying the arch types referred to by the element. Thus, the icon of the Customer (of the arch type Party) is a stick man.

**Containment:**

As the name suggest a strong degree of ownership is suggested which is why the contained object is graphically shown within the parent object and nowhere else. An object can be created externally and thereafter dragged into an object that is allowed to host it. This can also be achieved by extending the halo hand icon from a parent object to grab a child object to internalize it in the parent. Clicking the arrow on the contained object expands the entire object, which will act just as an external “normal” object would with the only difference of being contained within another object. A contained object can easily be externalized again simply by dragging it outside the parent container. In figure 3, the red object class, SalesOrderLine, is contained within the SalesOrder.

**The halo:**

We chose to visualize just the elements common to all Business objects in the halo: the relation (master/detail) and containment elements since displaying them all would work against overview and clarity. A hand halo element (the paw halo symbol on figure 3) is extended from the parent object to grab a child object that the developer which to have contained in the parent. This will have the effect of placing the child inside the parent Furthermore, notice that the design relies strongly on opening up elements within the business object, which relieves the halo form some of its options. Dragging a relation master or detail to the canvas itself presents a creation menu, which contains the known business object classes.

**Zoom facilities**

As mentioned, the overview is extremely important especially when modelling larger solutions. To facilitate overview, the system will have the following two features:

1. Split-pane view: It should be possible to have a split-pane view of the solution such that one pane would give the overview and the other would allow for working in detail.
2. Zoom-panel: A feature used in several applications that allows the developer to navigate by clicking in a minimized model resulting in navigation on the large model. Examples are the Knight Project [6] and Photoshop.

## 4 Intentional programming

The traditional way of implementing solutions like the one discussed in this paper is through the construction of an application framework supplying a set of tools for modeling a certain domain thereby giving the domain a means of expression and a language [3, p.2] – this is the way our current solution has come into being and it is along those tracks the solution continue to evolve as well.

Although a bit diffuse and unproven the ideas of intentional programming (IP) seems very promising and the basic ideas very attractive for domain specific modeling offering freedom of notation (even graphical notations), domain specific output from debugging, and a range of other benefits as well [1, p.503]. Charles Simonyi, has in a
Microsoft project constructed an IP environment with these desirable features [1] but it is unknown what has become of it after the project was withdrawn.

4.1 What is intentional Programming?

IP is an extendible programming environment based on active source [1, p.503], which means source with behavior at programming time. Extension libraries are loaded into the environment allowing for domain specific programming language features (an extension library can be for a general purpose language such as C or Java) [1, p.504]. The intentional extension libraries may extend any part of the programming environment from debugger, compiler to editor etc.

The idea of IP is to make abstractions self-describing so that they know what to do and how to display themselves on their own [2]. The important point is to capture the original computational intent of the programmer – how it is displayed, implemented etc. is secondary to the intent and is likely to change over time [2] which is why it is a great idea to have the intentions in a separate layer – as a plug-and-play component as de Moor puts it [3].

By writing methods operating on the source-graph [1, p505] intentions are able to display themselves, know how they should be debugged etc.

Since useful abstractions are really the core of domain specific languages they could benefit immensely from IP.

The source code representation is Abstract Syntax Trees (AST). Each node in the AST has a link to its declaration, which is why programs are represented as source graphs (interlinked source trees) [1, p.512].

The code written does not have to be parsed because it is in the format of a syntax tree with links to declarations and compilation is rather referred to as reduction which means that the code through extension libraries is transformed into platform specific code such as i.e. Java bytecode [1, p.511]. This makes the environment fast and responsive.

Intentions live one level above object classes: If abstractions are implemented as intentions rather than object classes (the class construct can be implemented as an intention) the abstractions will actively support editing, compiling, testing etc. [1, p.505] as opposed to the nature of regular, static, text based classes. Intention libraries operate on the source graph at design time as well making the source active.

The foundation of IP, AST, has proved its usability with respect to features very useful in domain specific modeling. The Mjoelner project [5] is an example of a strictly AST based environment where the representation of a source is just an arbitrary kind of pretty printing of the AST. To a large extent this feature also seems to be one of the major possible benefits of IP.

Domain specific attributes are in our present design presented and configured in the same manner. The idea from IP would be that each domain specific attribute contains its own notion of presentation and configuration. E.g. an address element could present itself with looks and functionality of a map, a company could be represented by the company logo, a Price element with looks and functionality of calculator etc.

There is a close correspondence between intentions and the predefined elements in the underlying ERP system. The elements capture important abstractions from the domain of ERP systems, and expose the variability through properties. However, it is
clear that the correspondence can be further elaborated, especially we can see that the different element types can have different graphical appearance, and allow modification of these properties in different ways. In our design, we already have distinctly different presentation of relationships, containment and other elements. But with the outset in IP, this idea can be pursued further. Presently double-clicking an element will open up a standard view of its properties, but it might presumably be interesting to allow deadline elements to present themselves with a build-in calendar or time line.

5 Discussion

Common UML CASE tools mostly share the trait that they have a relatively simple graphical user interface with a rather “square” feel to it. UML models have to be very abstract because there is no way to know beforehand what the user has in mind for his model – everything is open ended. Our solution is domain specific and elements are known beforehand allowing a much more specialized graphical expression as well as customized graphical features relating to domain specific traits. The main difference between a traditional UML diagram and our representation is that our model contains attributes of a domain specific nature and also represents the special relationship type “Containment”.

Moving the properties of objects onto the objects themselves is a reasonable thing to do since this is where they logically belong. Having the configurable elements in either a drop down menu or in a halo living on the object to be configured strengthen the mental model of the domain and assist the developer in the configuration process. It makes it easier to realize which objects are allowed in a configuration context and thus (hopefully) affects the learning curve of the configuration process. It also eliminates the need to shift focus from one spot to another in situations where focus is relevant on the object in question.

In the absence of intentional programming, AST is an attractive substitute for several reasons and have been offering many of the advantages now being associated with IP:

1. It is grammar based and is thus able to guaranty that any state of the program/environment/configuration will be legal.

2. Different tools share the same foundation so that information isn’t repeated and no inconsistencies arise on that account.

3. Since AST offer a shared program representation and even the storage structure of an application may be AST, different representations are basically pretty prints of this structure.

4. Browsing and hyper linking is facilitated with this kind of tree structure (which is a beneficial trait in graphical representations).

That is: AST’s can be a powerful means to implementing domain specific features just like IP could be.

Intensions would have the additional benefit over AST of isolating the domain specific abstractions in separate layers, independent of implementation language etc. which would be a welcomed feature in times of fast change. IP, just as well as AST, offer pretty printing methods operating on the source graph, and thus allows graphical notations that will have the same validity as any other form of notation.

Currently, the UML inspired diagrams as presented in this paper has not been implemented. However, the underlying model is in place.
6 References


