NEXT: Generating Tailored ERP Applications from Ontological Enterprise Models

H.W. van der Schuur[1], E. van de Ven[1], R. de Jong[1], D.M.M. Schunselaar[2], H.A. Reijers[2], M. Overeem[1], M. de Graaf[1], S. Jansen[3], and S. Brinkkemper[3]

[1] Department of Architecture and Innovation, AFAS Software, The Netherlands
hwschuur@gmail.com,{e.vdven,r.dejong,m.overeem,m.degraaf}@afas.nl
[2] Department of Computer Science, Vrije Universiteit, The Netherlands
{d.m.m.schunselaar,h.a.reijers}@vu.nl
[3] Dept. of Information and Computing Sciences, Utrecht University, The Netherlands
{s.jansen,s.brinkkemper}@uu.nl

Abstract. Tailoring Enterprise Resource Planning (ERP) software to the needs of the enterprise still is a technical endeavor, often requiring the (de)activation of modules, modification of configuration files or even execution of database queries. Considering the large body of work on Enterprise Modeling and Model-Driven Software Engineering, this is remarkable: Ideally, one models one's own enterprise and, at the press of a button, ERP software tailored to the needs of the modeled enterprise is generated. In this paper, we introduce NEXT, a novel model-driven software generation approach being developed with precisely this goal in mind. It uses the expressive power of ontological enterprise models (OEMs) to generate ERP cloud applications. An OEM only describes the real-world phenomena essential to the enterprise, using terms and customizations specific to the enterprise. We present our considerations during development of the OEM modeling language, which is designed to capture the specifics of enterprise phenomena in a way that technical details can be derived from it. We expect NEXT to drastically shorten the time-to-market of ERP software, from months–years to hours–days.

Key words: ontological enterprise modeling, model-driven software development, software generation, enterprise resource planning

1 Introduction

ERP software has become fundamental in the day-to-day operation of enterprises. Although most ERP applications provide functionality to tailor the software to the specific needs of the enterprise, generally, customization possibilties are limited and require technical knowledge from the end-user. As an example, a fixed number of free input fields, input records or modules can be activated or deactivated, for instance through advanced management tools, specific configuration files or by executing particular database queries. As a result, enterprises have to tailor their processes to the software, instead of the software being tailored to the enterprise. This is surprising given

⅄ This work is a result of the AMUSE project. See www.amuse-project.org.

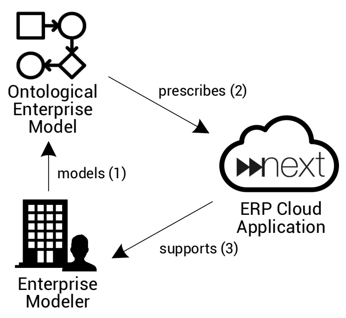


Fig. 1: The NEXT software generation approach

the comprehensive bodies of work on Enterprise Modeling and Model-Driven Software Engineering. Ideally, one models one's own enterprise and, at the press of a button, ERP software tailored to the needs of the modeled enterprise is generated.

In this paper, we show preliminary results obtained during the ongoing development of NEXT. NEXT is a novel software generation approach using ontological enterprise models (OEMs) to generate ERP cloud applications. With NEXT, functional application requirements are separated from the application logic and the technical foundations of the application: all enterprise-specific requirements are expressed in the OEM through one modeling language, allowing for generation of a functionally tailored ERP application. This work is a first effort to describe the many aspects and ideas encompassing our approach (see Figure 1). First, we introduce NEXT's declarative OEM Language and present our considerations during its development. The OEM Language allows the modeler to reason and model in terms of real-world enterprise concepts, without having to consider technical implementation details (arrow 1 in Figure 1). The language is designed such that an OEM forms the basis for generating an integrated ERP cloud application (arrow 2). As a result, the generated application is tailored to the needs of the modeled enterprise by definition (arrow 3).

This paper is structured as follows: in Section 2, NEXT's OEM Language is outlined. In Section 3, we illustrate the model transformation and software generation processes through three concrete models, and show the generated ERP cloud application for each of these models. Finally, we compare our approach with similar enterprise modeling and model-driven software generation initiatives in Section 4, and present conclusions and future work in Section 5.

2 The Ontological Enterprise Modeling Language

In this section, we describe the OEM Language as well as our considerations during the development of the language.

During the past decade, being ignorant of existing Enterprise Modeling (EM) languages, the OEM Language has been developed within AFAS. Recently, we have started evaluating existing EM approaches for their applicability to automatically generate ERP software and none sufficed [1]; also see Section 4. Next to this, from a pragmatic side, by developing our own language we have full flexibility to (re)define the language to best fit our needs (see Figure 1 and Section 2.2). At the same time, we acknowledge that a more thorough analysis is required to further justify the existence of another EM language; this analysis is subject of future work.

In the remainder of this section, we start with an exemplifying model to familiarize the reader with the language and its main constructs. Next, we present our considerations during development of the language.

2.1 EnYoi: An Example Enterprise

Within this example, we suppose there is an enterprise named EnYoi. EnYoi's core business is selling shaving products. Its main customers are hotels. Products are sold both ad hoc and through subscriptions. From time to time, EnYoi sends free trial products to its customers to find out if there is substantial interest in the products to bring them to market. Already from the description of EnYoi, we can identify concepts that are relevant within EnYoi, e.g., selling, products, subscriptions, customers. For particular concepts, one would expect specific application functionality, e.g., when an enterprise sells physical products, the enterprise needs an application to register orders, maintain stock levels, and obtain actual insights in the (historical) economical performance of the enterprise. The identified enterprise concepts of EnYoi map onto the four main stereotypes in the OEM Language: Entity, Role, Event, and Agreement. Using our example enterprise EnYoi, we elaborate on each of them.

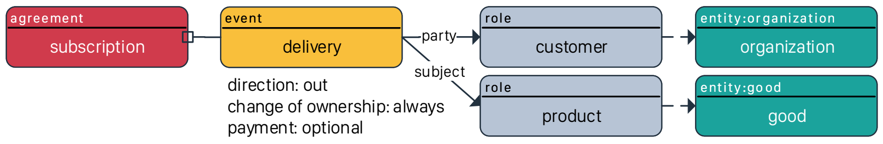


Fig. 2: A simple OEM used by EnYoi

In Figure 2, we have the OEM of EnYoi. The hotels EnYoi is delivering to are denoted with the block *organization*. Similarly, what EnYoi is delivering are goods (denoted by the block *good*). The organization block and good block are both of the type **entity**. Entities are things in the real world around us, e.g., time, locations, people, organizations. Furthermore, entities have a set of universal, constant properties, i.e., all physical things (goods, persons, etc.) take up space, are tangible and have weight. To capture the semantics of an entity, each entity is of a particular type. Currently, the three most mature types of entities for which we have implemented specific functionality in the application are: organization, person, and good. Using the type of an entity, we deduce common-sense[1] functionality. For instance, for the organization type, we generate

[1] We use the term 'common-sense' in line with the work of Fox [2], i.e., information that is deduced from the semantics of the model.

functionality to administer, i.a., contact persons, date of establishment, date of closing down, email address (all common-sense in our ERP domain). Analogously, the entity type person entails functionality related to, i.a., date of birth, sick leave, scheduling. Finally, for the entity type good, functionality is generated related to, i.a., physicality (size, location, weight), value, ownership. All common-sense functionality comes for free, i.e., *without the need to model this explicitly*[2]. At the time of writing, we already have additional types in scope such as location, time, room and country, which will be detailed in future work.

Next to entities, we have **roles**. Roles allow the modeler to denote in what way entities are considered from within the enterprise. For instance, for EnYoi, hotels perform the role *customer*. Analogously, some goods perform the role of *product*. Note that not all goods within EnYoi need to be products, e.g., forklift trucks used to move pallets with shaving products will typically not be a product but it is a good. Entities can perform multiple roles from the perspective of an enterprise, e.g., some organizations might be both customer and supplier.

As mentioned, EnYoi delivers products to its customers. The *delivery* of products is modeled using an **event**. Events are used to abstract real-world business activities that are relevant for the enterprise to administer. Although at design-time an event is a static, timeless component, during run-time, an event is executed at a particular moment in time. The potential run-time effects of an event (e.g., stock level decrease) are encoded using *characteristics*. By combining characteristics, we know what an event means, i.e., the meaning of an event is determined by the combination of the meanings of the individual characteristics. Based on the characteristics of an event, we add common-sense to the model. Using the characteristics *subject* and *party* of an event, depicted on the edges between events and roles, we know how these roles and their entities are involved in the event. For instance, the subject denotes what this event is about, e.g., *what* is being delivered, and the party is another stakeholder in this event. Again, within EnYoi, products are delivered to customers.

Some enterprises sell goods, and some purchase goods. To distinguish between selling and purchasing, we have introduced a characteristic *direction* on the events. By setting the direction to *in* (*out* respectively), we indicate that something of value enters (leaves respectively) the enterprise. Next to selling and purchasing goods, enterprises can also rent (or rent out) goods. In the former case, there is a change of ownership, and in the latter there is not. This resulted in a *change of ownership* characteristic. Next to administering that an enterprise obtained the ownership of a good, the enterprise also differentiates in what capacity they own something, which is supported in the *ownership type* characteristic; currently trade item and asset are supported. The direction, change of ownership, and ownership type characteristics specify the (financial) effects on stock and the general ledger accounts, e.g., if the enterprise obtains the ownership of an asset (direction in), then periodic deprecations of the asset is journalized in the general ledgers. Furthermore, if the enterprise loses the ownership of a trade item (direction out), then the stock levels will be decreased. In the EnYoi example, for some deliveries a payment was expected and some were free of charge. This can be indicated using the

[2] Every stereotype can also have modeled (non-semantic) attributes, e.g., description, number of employees, wedding date, margin, or attachment.

payment characteristic on an event, i.e., is there a monetary claim created to the party after executing this event. By setting the payment characteristic to *optional*, the generated application will give the user the run-time choice to create a monetary claim for every event instance. Based on the payment characteristic, we deduce common-sense that includes, i.a., accounting functionality to give insights into the ledger accounts and trial balance (see Section 3).

The last model element in Figure 2 is the *subscription* which has type **agreement**. This indicates that deliveries can be performed within the context of an agreement. An agreement in itself only represents the fact that there is an agreement with a second party. The contents of the agreement are represented by events that are connected to the agreement (e.g., delivery for EnYoi).

2.2 Language Considerations

As shown in Figure 1, an enterprise modeler models the OEM using the OEM Language (1). This OEM prescribes an ERP cloud application (2). In its turn, this application supports the enterprise (3). For each of these arrows, we present our challenges in the form of questions we posed ourselves and our considerations in development of the OEM Language in the remainder of this section.

(1) An Enterprise Modeler models an OEM

What is our target audience? Current ERP software requires people with substantial technical knowledge to configure and tailor the software towards the enterprise needs. Often these people are less familiar with the enterprise itself. As a result, considerable communication is required between people familiar with the enterprise and the more technical application managers. Within NEXT, we have targeted the enterprise modeler familiar with the enterprise as primary audience for the OEM Language. Enterprise modelers are not necessary familiar with technical concepts like foreign keys, database tables, and validations, but are familiar with the people, products, processes, and other concepts within the enterprise. To better connect to the target audiences, we have stereotypes and characteristics in the OEM Language that are named after the real world phenomena they are representing, e.g., agreement, or payment. Please note that the models we are showing are *a* possible visualization. The usability of language by end users is still subject of future investigation. For the first release of NEXT, all the OEMs for enterprises will be created by modelers from AFAS.

What should be expressible within our language and how? As mentioned in the introduction, we want to separate the functional application requirements from the technical foundation of the application. As a result, our OEM Language should only contain enterprise concepts; it should not contain IT artifacts. To determine which enterprise concepts should be part of the OEM Language, we continuously analyze AFAS' current ERP application named Profit, which is currently used by more than 1.3 million end-users of 10.000 customers. By analyzing the (usage of) functionality provided by Profit, we verify that all enterprise aspects of the current set of customers can actually be expressed in the OEM Language. This analysis is conducted by processing all fields, screens, database tables, etc. of Profit. For each field, screen, database tables, etc., we determine with which functional reason it was introduced. Furthermore, we determine if there is variability of the particular functionality between enterprises, i.e., which functionality is used differently by the customers, e.g., the moment an order is considered finalized differs per enterprise. If functionality is used differently between customers, we investigate what the reason behind this is. Depending on the reason, the level of variation, and the number of customers using a particular variation, we decide whether to encode this variability in the OEM Language by means of characteristics. Finally, based on this analysis, we determine if, and how, the functionality the screen, database table, etc. represents should be part of the OEM Language. We determine this as follows: If there is no variability in a particular part of the functionality among enterprises, then this is not explicitly represented in the OEM Language (this part of the functionality will be generated without modeling it). If there is variability, then we try to map this onto real world phenomena. If such a mapping exists, then the real world phenomenon will be part of the languages, e.g., change of ownership, agreement. So far, we did not encounter any variability in functionality where this mapping to the real world phenomena did not exist. Using this mapping onto real world phenomena, we have created our characteristics and stereotypes.

If all functionality provided by Profit can be modeled using the OEM Language, then the first version of the OEM Language is considered complete. As a result, the OEM Language will, initially, be scoped to the current functionality of Profit. In later phases, this scope is widened for every new release of Profit. Also, when required, the OEM Language will be extended with additional concepts to support functionality (currently) not supported within Profit.

(2) An OEM prescribes an Application
How to ensure every OEM *is complete and valid input for generating exactly one application?* The goal of an OEM is to prescribe an application tailored towards the needs of an enterprise. As a result, given an OEM, it should always be complete and valid input for generating an application. Furthermore, there cannot be two functionally different applications adhering to one and the same OEM. This would mean that the enterprise modeler cannot anticipate what functionality is provided in the generated application, which is highly undesirable for obvious reasons. To ensure that there cannot exist two functionally different applications based on one and the same OEM, we are formalizing the language. This way, there cannot be any ambiguity with respect to which functionality should be generated based on a particular OEM. To ensure that every OEM is complete and valid input for generating an integrated, fully-functioning application, every stereotype is allowed to exist independently within a model, i.e., an OEM that consists of one stereotype in isolation is already complete and valid input for generating the application. Furthermore, stereotypes can only be composed if the composition maintains the completeness and validity of the OEM. As a result, every OEM is complete and valid by construction.

How to keep the language maintainable when concepts change over time?
The environment around an enterprise is always evolving, e.g., rules and regulations change, distribution channels change, more innovative competitors appear. As a result, the OEM Language also needs to evolve, e.g., maternity leave, sick leave, holiday, etc. are concepts that did not always exist. When concepts change, or are introduced, one ideally only changes the stereotypes representing these concepts in the language, i.e.,

ideally the changes are local. We attempt to achieve these local changes by requiring that the composition of stereotypes only adds behavior and does not change existing behavior. As a result, if a concept changes, we only need to change the stereotypes and compositions representing the concept. Would we not have local changes, then a change of a concept would require a change to the stereotypes and compositions representing the concepts, as well as, stereotypes and compositions affecting the stereotypes and compositions representing the concepts.

(3) An Application supports the Enterprise

How to support variability within enterprises? Not a single enterprise is exactly identical to another enterprise. To capture the variability between enterprises, we have introduced the characteristics on events (Section 2.1). Also variability exists *within* an enterprise, e.g., in the case of EnYoi, some deliveries require a payment and some are for free. This internal variability is encoded within the OEM Language by offering the enterprise modeler the possibility to postpone setting characteristics till run-time. This means that for every new run-time instance, the characteristics can be set differently.

How to provide the same run-time support as Profit? Profit provides run-time support related to: workflow, rules, Key Performance Indicators (KPIs), authorization, customer portals, etc. All this run-time functionality should be provided using the OEM. After all, the OEM prescribes the applications. In current approaches, these KPIs are defined by hand. Often they also need to be tailored towards a particular enterprise. In order to automate this manual endeavor, we have created deduction rules, which, given an OEM, automatically can deduce the KPIs and present them and their values during run-time. The other functionality, e.g., workflow, is still under investigation. Particularly, if and how the functionality can be deduced from an OEM and which information is still missing in the OEM Language to automatically deduce this functionality with its possible variability.

3 From Design-time to Run-time

How NEXT cloud applications are to be generated and deployed is depicted in Figure 3 (arrows labeled with numbers correspond to the arrows in Figure 1). Starting on the ontological layer, the real-world enterprise is abstracted by an OEM. OEMs are expressed in the OEM Language using our modeling tool called Studio. Studio understands the OEM Language as well as the common-sense (see Section 2) that is part of it. The resulting OEM serves as input for the cloud application generation process, which is orchestrated by the NEXT Cloud Manager. The Cloud Manager's responsibilities are fourfold:

First, the Cloud Manager reads the OEM and transforms it to a Platform-Independent Model (PIM), which contains high-level application constructs that establish the required run-time functionality. Next, it uses the PIM to create a Platform-Specific Model (PSM) that forms the basis for the actual software generation process. Secondly, the Cloud Manager configures the cloud infrastructure. The cloud infrastructure consists of (1) a cloud computing platform such as Microsoft Azure, Amazon Web Services or Google Cloud Platform and (2) cloud services on top of the cloud computing platform

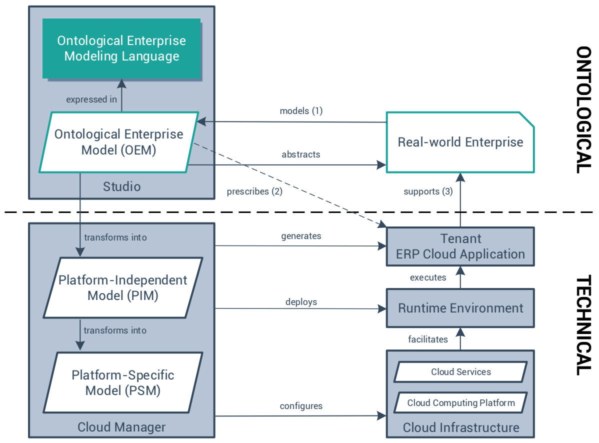


Fig. 3: Model-Driven Software Generation with NEXT

such as application hosting services, event buses or database services. The Cloud Manager determines the type and amount of services required. Thirdly, the Cloud Manager deploys the runtime environment in which the tenant ERP cloud application is executed. For example, generic frontend and backend framework code used by the cloud application is deployed. Also, generic runtime services such as an authentication service and logging service are deployed. Finally, the tenant ERP cloud application itself is generated. The application is generated based on the PSM, which was created earlier from the PIM (and indirectly, the OEM). After the Cloud Manager has completed its task, a fresh NEXT ERP cloud application tailored to the modeled enterprise is generated and ready for use by that enterprise. Many technical details involving our software generation approach are omitted to conserve space, and are detailed elsewhere[3].

The remainder of this section describes how an OEM transforms to a corresponding PIM and onwards to a PSM. In three steps, we build the OEM of the EnYoi example enterprise from Section 2. See Figure 4. Each of the columns depicts one particular OEM (top 'OEM' layer), as well as the transformation to the corresponding PIM and PSM ('PIM' and 'PSM' layers). We will start with a simple OEM on the left and expand the model as we go from left to right. This is done in such a way that a model in a particular cell *c* is a fragment of the model in the cell to the right of *c*. Every OEM is self-contained: an OEM in itself is sufficient to generate an application from. The cells in the PIM and PSM rows are complementary, i.e. PIM/PSM elements in cell *c* should also be considered to be part of every cell to the right of *c*. The colors of the blocks with black text in the OEM and PIM layers denote a causal relationship: colored rectangular blocks in a particular column's cell *c* are the result of the rectangular blocks with the

[3] For example, NEXT uses Event Sourcing [3] to ensure that all changes to the application state are stored as an event sequence. Using this sequence of events, application-wide features such as auditing, logging of in-the-field software operation and usage [4] as well as application rollbacks are implemented [5, 6].

same color(s) in the cell above *c*. For example, the green elements Organizations and Goods in columns ❶ and ❸ in the PIM layer result from the green organization and good entities in the OEM layer, respectively. Below, the transformations in each of the columns ❶–❸ are described in more detail.

❶ An Organization Performs a Customer Role...

The first OEM is composed of a customer *role* which is performed by an organization *entity*. The design-time entity organization results in functionality to create, read, update, and delete (CRUD) organization instantiations in the PIM. As mentioned, the fact that organizations can have contact persons is considered common-sense in our ERP domain. CRUD is therefore *also* generated for contact persons. In addition, functionality is generated to assign contact persons to organizations. Similarly, because of the customer role in the OEM, the role customer can be assigned to an organization. Furthermore, generic application functionality is generated irrespective of the source OEM. This is represented by the PIM level Runtime Framework block. The runtime framework establishes basic navigation (including pages, lists, forms, etc.), search, and reporting functionality for each application.

Next, all PIM blocks (CRUD, Runtime Framework) and their elements are transformed into technical counterparts in the PSM. The PSM consists of code generator templates for both the front end and back end of the NEXT runtime[4]. For example, the CRUD block is translated into code generator templates to accommodate the runtime creation, presentation, revision and deletion of organizations. Using the same techniques, the Navigation, Search and Reporting functionality of the Runtime Framework are transformed to the PSM level.

❷ ...in a Delivery Event...

In column ❷, the OEM is further extended with three elements. First, the delivery *event* is added. Secondly, by assigning a role to the event's subject characteristic, one models what is going to be delivered by the modeled enterprise. In the OEM of column ❷, goods that perform the product role can be subject of the delivery event.

The extension of the OEM with the delivery event, the product role, and the good entity results in additional PIM elements. First, CRUD functionality specific to the good entity is added, including functionality to assign the product role to existing goods. Furthermore, CRUD functionality is added specific to the delivery event so that new deliveries of products to customers can be created.

As mentioned in Section 2.1, run-time event instantiations are temporal by definition, i.e. an event is always executed at a particular moment in time. Also, events have various characteristics which can be set to configure the precise ontological meaning and resulting run-time behavior of the event. If there is at least one event in the OEM with {subject: good; change of ownership: yes; direction: out} characteristics, Stock logic is generated on the PIM level: if there are enterprise events that move goods (i.e.,

[4] At the time of writing, we have front end code generator templates for HTML, Javascript, and CSS. The back end generator templates currently generate C# code.

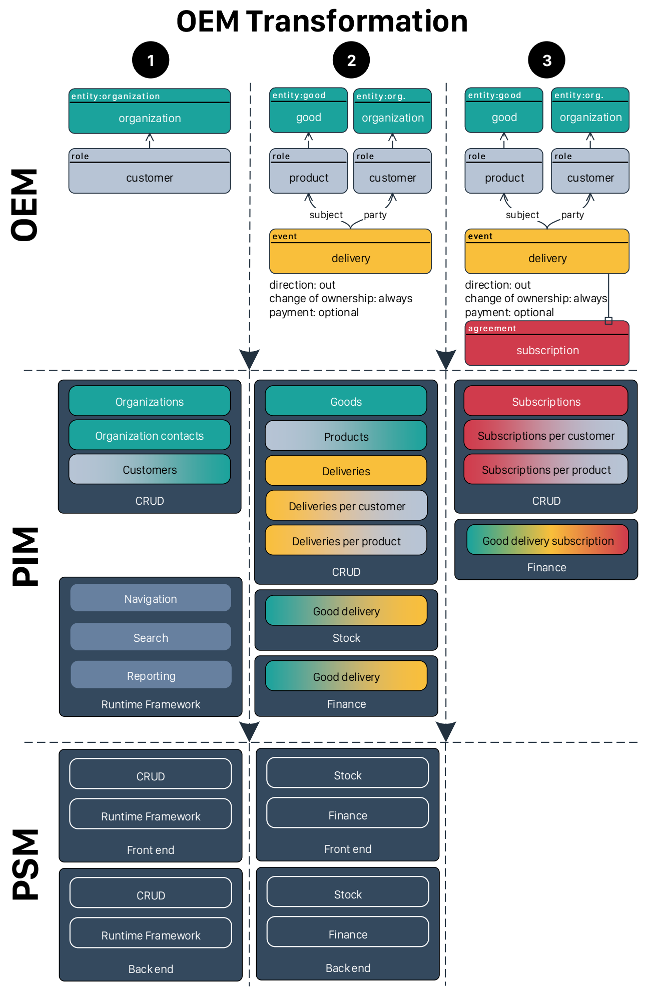


Fig. 4: Three OEMs and their transformations to corresponding PIMs and PSMs. Each OEM cell is independent, whereas the PIM and PSM cells also implicitly include the cells to the left. All OEMs result in a functioning cloud application (see Figure 5).

a physical, tangible, valuable object) outwards, it is plausible that the enterprise would like to have insight in the (remaining) stock levels of the particular good. Analogously, the Finance element is generated on the PIM level if there is at least one event in the OEM with characteristics {subject: good; payment: yes}: if one expects to receive or perform payments, one would also like to obtain insights in financial journal entries, revenue, profit and loss account, the enterprise's trial balance, etc. Note that if there is variability in the stock logic, e.g., deliveries without a change of stock, then this variability should be reflected in the OEM Language, e.g., by means of characteristics.

On the PSM level, for both the front end and back end, additional code generator templates are instantiated to generate the code required for the additional Stock and Finance functionality from the PIM level.

❸ . . . According to a Subscription Agreement

Finally, in column ❸, a subscription *agreement* is added to the OEM. The delivery event can now be part of an agreement: as opposed to delivering in an ad hoc and impromptu fashion (column ❷), the delivery event can now be the consequence of, and governed by, a subscription agreement. The subscription forms an agreement in which delivery concerns such as the delivery frequency, the amount to be delivered, subscription duration, etc. are established. A single agreement can govern multiple events, e.g. delivery, invoicing, and payment.

Again, the addition of a new OEM element results in an extension of the PIM. First, CRUD functionality specific to the subscription agreement is added, including functionality to assign subscriptions to existing customers. Also, the Finance block that was added to the PIM in column ❷, is extended in the PIM of column ❸ because of the addition of the subscription element in the OEM. Since the delivery can now be governed by the subscription, both the outgoing value and the expected revenue can be calculated within the subscription context and period. Additional financial overviews, such as an operations overview and operations forecast, are generated in the PIM.

On the PSM level, no additional elements are introduced: all PIM elements from column ❸ are expressed with the PSM elements in columns ❶ and ❷.

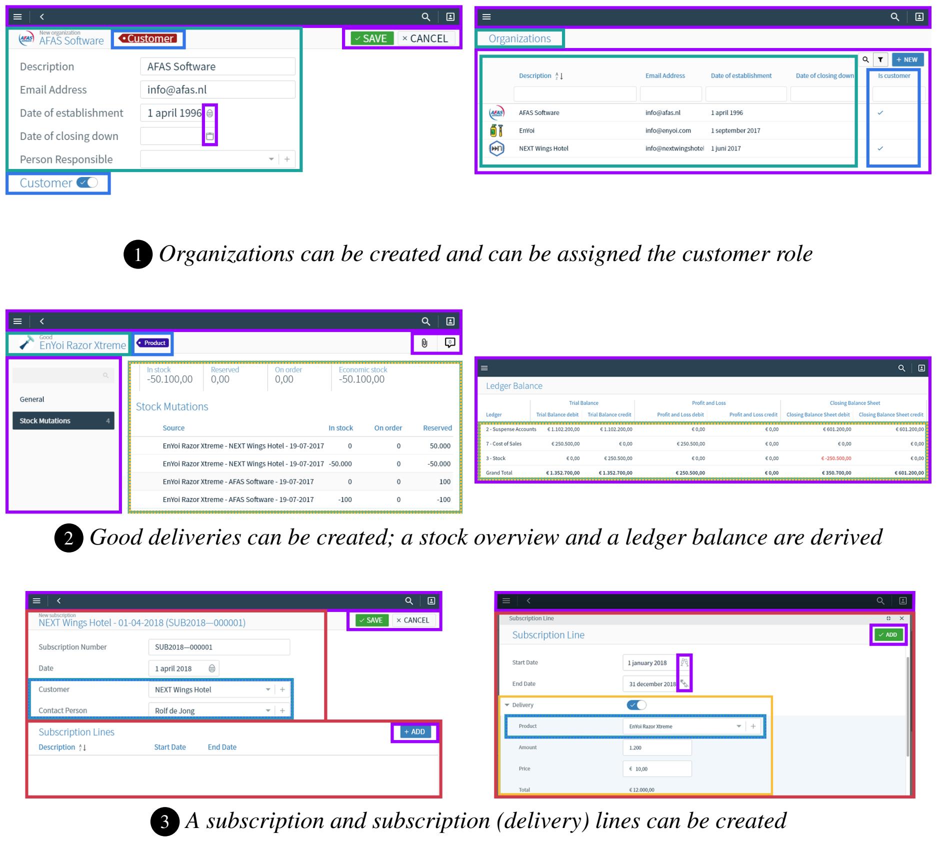
Current State of the Runtime

To show the current state of the NEXT runtime, Figure 5 shows run-time screenshots of generated NEXT cloud applications for each of the OEMs in Figure 4. The application of ❶ is rather basic and straightforward: organizations can be created and listed, and they can be assigned a customer role (note that already with application ❶, basic functionality such as navigation and search is available). With the application generated from ❷, good deliveries can be created. Based on run-time deliveries, stock mutations, a stock overview as well as ledger balances are automatically derived. Finally, with the application based on OEM ❸, delivery subscriptions can also be created. Note that the start and end date, as well as the subscription duration are automatically derived.

At the time of writing, NEXT itself consists of more than 580 KLOC and the most substantial OEMs currently generate 1.2 MLOC.

4 Related Work

With NEXT, we connect the worlds of enterprise modeling (EM) and model-driven software engineering. Next to this, work has been done on creating models of ERP functionality. We provide related work from all three domains.



Description	AFAS Software
Email Address	info@afas.nl
Date of establishment	1 april 1996
Date of closing down	
Person Responsible	

	Description	Email Address	Date of establishment	Date of closing down	Is customer
	AFAS Software	info@afas.nl	1 april 1996		✓
	EnYoi	info@enyoi.com	1 september 2017		
	NEXT Wings Hotel	info@nextwingshotel	1 juni 2017		✓

1 *Organizations can be created and can be assigned the customer role*

In stock	Reserved	On order	Economic stock
-50.100,00	0,00	0,00	-50.100,00

Stock Mutations

Source	In stock	On order	Reserved
EnYoi Razor Xtreme - NEXT Wings Hotel - 19-07-2017	0	0	50.000
EnYoi Razor Xtreme - NEXT Wings Hotel - 19-07-2017	-50.000	0	-50.000
EnYoi Razor Xtreme - AFAS Software - 19-07-2017	0	0	100
EnYoi Razor Xtreme - AFAS Software - 19-07-2017	-100	0	-100

	Trial Balance		Profit and Loss		Closing Balance Sheet	
Ledger	Trial Balance debit	Trial Balance credit	Profit and Loss debit	Profit and Loss credit	Closing Balance Sheet debit	Closing Balance Sheet credit
2 - Suspense Accounts	€ 1.102.200,00	€ 1.102.200,00	€ 0,00	€ 0,00	€ 601.200,00	€ 601.200,00
7 - Cost of Sales	€ 250.500,00	€ 0,00	€ 250.500,00	€ 0,00	€ 0,00	€ 0,00
3 - Stock	€ 0,00	€ 250.500,00	€ 0,00	€ 0,00	€ -250.500,00	€ 0,00
Grand Total	**€ 1.352.700,00**	**€ 1.352.700,00**	**€ 250.500,00**	**€ 0,00**	**€ 350.700,00**	**€ 601.200,00**

2 *Good deliveries can be created; a stock overview and a ledger balance are derived*

Subscription Number	SUB2018—000001
Date	1 april 2018
Customer	NEXT Wings Hotel
Contact Person	Rolf de Jong

Subscription Lines

Description	Start Date	End Date

Start Date	1 january 2018
End Date	31 december 2018
Delivery	✓
Product	EnYoi Razor Xtreme
Amount	1.200
Price	€ 10,00
Total	€ 12.000,00

3 *A subscription and subscription (delivery) lines can be created*

Fig. 5: Screenshots of NEXT applications based on the three OEMs in Figure 4. UI elements are annotated with the color of the particular stereotype(s) causing generation of the elements, i.e., green: entity (organization, person), blue: role (customer, product), yellow: event (delivery), red: agreement (subscription). Purple indicates generic application functionality that is generated irrespective of the underlying OEM. Full-size versions can be downloaded from www.amuse-project.org/portal-amuse/nextruntime.

4.1 Enterprise Modeling

In our previous work [1], we compared EM approaches on their applicability for automatically generating ERP Software. These approaches included, among others: ArchiMate [7], ARIS [8], CIMOSA [9], DEMO [10], MEMO [11], MERODE [12], and UEML [13]. We show some of the highlights of [1] in the context of this paper. We do so, by going through the steps listed in Figure 1 and show the requirements on the EM approach. We omit step (3) since none of the approaches are applicable to automatically generate ERP software [1].

(1) An Enterprise Modeler Models an OEM

One aspect on which NEXT sets itself apart from existing EM approaches is the ontological aspect [1]. With this, we do not mean that existing approaches do not have ontologies, but that the NEXT ontology is on a different level; it has more detail. For example, within DEMO [10], there are so-called transactions. The ontological characteristics of a transaction is that it represent a set of steps, i.a., request, promise, and accept. Within DEMO, the transaction itself only has a label associated to it. In NEXT, we aim to add more information, e.g., within *events* (Section 2.1), we have characteristics to detail the semantics of a particular event and not only the steps every event goes through. If we would translate the ideas from NEXT to DEMO, this would mean that more information is known about transactions, e.g., whether it entails the sale of an item, or borrowing a book. Similarly, in the MEMO approach [11], there are processes. These processes do have an ontological distinction between automatic, semi-automatic, and manual. But apart from this, processes have a label with no further semantics for the application. Another differentiating aspect between NEXT and some EM approaches is the targeted audience. Within some approaches, information needs to be provided in a programming-like style. In NEXT, we take the business user as a target audience. This target audience is usually not familiar with programming. If we take the DEMO approach again [10], then within DEMO there is a so-called *action model*. The action model specifies the action rules that serve as guidelines for the actors. The specification of an action model is syntax-wise very close to the syntax of a programming languages. Other approaches where some aspects of the approach require a user with programming skills include [1]: ARIS [8], CIMOSA [9], and MERODE [12].

(2) An OEM Prescribes an Application
One of the most essential parts for describing the application from an enterprise model is formal (execution) semantics. After all, without clear semantics, either functionally different applications would adhere to the same enterprise model, or it is not possible at all to generate an application. Most of the EM approaches have formal (execution) semantics [1]. Some, like ArchiMate [7], CIMOSA [9], DEMO [10], and UEML [13], are not completely formally defined with respect to (execution) semantics [1].

4.2 Model-Driven Software Engineering

Kulkarni [14] reflects on decades of experience in delivering large business applications using a model-driven development approach, and concludes that models with a higher level of abstraction and expressive power lead to many advantages such as (a) more significant operational involvement of functional experts, (b) early determination and elimination of errors, (c) application-wide implementation of design decisions and policies, and (d) separation of the functional application specification from technology concerns. The NEXT ontological enterprise models encompass a high level of abstraction and we pursue these advantages. Since the OEM Language abstracts from technical elements such as window types, forms, and batch functionality (see Section 2), we believe it provides a higher level of abstraction than the high-level language Q++ developed by Kulkarni [14]. Existing model-driven enterprise engineering solutions such as Mendix, Servoy, or Betty Blocks[5] are not based on ontological models; these require

[5] www.mendix.com, www.servoy.com, www.bettyblocks.com

technical knowledge from the modeler to enable and influence architectural aspects such as storage, business logic, and frontend behavior [15]. While the aforementioned solutions allow for the creation of generic business applications which support communication, collaboration, and content creation between business functions, NEXT is specifically designed to model, understand and generate integrated ERP applications. The model-driven enterprise engineering solutions from IBM[6] and OpenText[7] specifically allow for the creation of business process management (BPM) applications and are often used on top of legacy software. Contrarily, NEXT is not designed to be used on top of legacy software; with the NEXT approach, integrated, fully-functioning ERP applications are generated.

4.3 ERP Modeling

Work has been done on creating models of ERP systems with the goal to configure an ERP system to the requirements an enterprise, e.g., [16]. Within these approaches, an abstraction is made of an existing ERP system by means of a model, a model is created of the customer requirements, and manually a mapping is made between both models to see how to configure the ERP system. Our approach sets itself apart from these approaches in a number of ways: (1) similar to existing EM approaches, the ontological aspect is limited, (2) we are aiming for an automated approach; no manual mapping, and (3) NEXT is not intended to be used on existing software systems.

5 Conclusions and Future Work

The paper is a first effort to describe the many aspects and ideas that encompass our NEXT software generation approach. We show that ontological enterprise models can form the basis for generating integrated, fully-functioning Enterprise Resource Planning (ERP) cloud applications.

We have presented the NEXT Ontological Enterprise Modeling Language, a language specific to the ERP domain. The language is being designed to provide sufficiently powerful semantics for modeling real-world enterprises, i.e., express concepts such as an enterprise's people, products, and business processes. Every model created with the language forms the basis for an integrated, fully-functioning ERP cloud application, tailored to the needs and requirements of the modeled enterprise. The OEM Language is being designed for the modeler to easily comprehend and maintain the enterprise model, while requiring minimal technical knowledge. Apart from the language itself, we have presented our considerations during development of the language.

Through three exemplifying OEMs, we have illustrated the OEM transformation process, and how we are able to generate sophisticated run-time ERP application functionality and behavior from (the composition of) basic modeling language stereotypes. More complex OEMs will be subject of future work.

[6] www.ibm.com/software/products/en/business-process-manager-family

[7] www.opentext.com/what-we-do/products/business-process-management

As NEXT is under development, its OEM Language and software generation approach are further enriched, refined and evaluated continuously. Also, many technical aspects that encompass NEXT, such as (partial) code generation techniques, blue green deployment, and real-time monitoring, are considered out of the scope of this paper and are or will be detailed elsewhere [6].

NEXT is designed to allow for creation of tailor-made software using one toolset and modeling language. We expect NEXT to drastically shorten the average time-to-market of ERP software, from months–years to hours–days. Despite the developmental stage of NEXT, we hope that this paper sheds new light on the potential of (ontological) enterprise modeling.

Acknowledgements This paper was supported by the NWO AMUSE project (628.006.001): a collaboration between Vrije Universiteit Amsterdam, Utrecht University, and AFAS Software in the Netherlands.

References

1. Schunselaar, D.M.M., Gulden, J., van der Schuur, H., Reijers, H.A.: A Systematic Evaluation of Enterprise Modelling Approaches on Their Applicability to Automatically Generate ERP Software. In: (CBI) 2016, IEEE Computer Society (2016)
2. Fox, M.S.: The TOVE project towards a common-sense model of the enterprise. In: IEA/AIE. Springer (1992) 25–34
3. Fowler, M.: Event Sourcing. `https://martinfowler.com/eaaDev/EventSourcing.html` (2005) Retreived on 09-09-2017.
4. Schuur, H. van der, Jansen, S., Brinkkemper, S.: Reducing Maintenance Effort through Software Operation Knowledge: An Eclectic Empirical Evaluation. In: CSMR, IEEE Computer Society (2011) 201–210
5. Overeem, M., Spoor, M., Jansen, S.: The Dark Side of Event Sourcing: Managing Data Conversion. In: SANER. (2017) 193–204
6. Overeem, M., Jansen, S.: An Exploration of the It in It Depends: Generative versus Interpretive Model-Driven Development. In: MODELSWARD. (2017)
7. Open Group: Archimate 2.1 Specification. Van Haren Publishing (December 2013)
8. Scheer, A.W.: Aris–Business Process Modeling. 2nd edn. Springer (1999)
9. Vernadat, F.: The CIMOSA Languages. In: Handbook on Architectures of Information Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 251–272
10. Dietz, J.L.G.: Enterprise ontology - theory and methodology. Springer (2006)
11. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. SoSyM **13**(3) (2014) 941–962
12. Snoeck, M.: Enterprise Information Systems Engineering - The MERODE Approach. The Enterprise Engineering Series. Springer (2014)
13. Vernadat, F.: UEML: Towards a unified enterprise modelling language. International Journal of Production Research **40**(17) (2002) 4309–4321
14. Kulkarni, V.: Model Driven Development of Business Applications: A Practitioner's Perspective. In: ICSE Companion. (2016) 260–269
15. Fortuin, S.: Model Driven Engineering: Incipient Environments with Imperative Views. Master's thesis, Utrecht University (2016)
16. Rolland, C., Prakash, N.: Matching ERP system functionality to customer requirements. In: 5th IEEE International Symposium on Requirements Engineering. (2001) 66–75