Domain engineering for customer experience management

Innovations in Systems and Software Engineering (2022) 18:171–191
https://doi.org/10.1007/s11334-021-00426-2

S.I. : SOFTWARE AND SYSTEMS REUSE

Domain engineering for customer experience management

Imen Benzarti[1] · Hafedh Mili[1] · Renata Medeiros de Carvalho[2] · Abderrahmane Leshob[3]

Received: 21 April 2021 / Accepted: 30 October 2021 / Published online: 20 January 2022

Abstract
Customer experience management (CXM) denotes a set of practices, processes, and tools, that aim at personalizing a customer's interactions with a company around the customer's needs and desires (Walker in The emergence of customer experience management solutions, 2011). The past few years have seen the emergence of a new generation of context-aware CXM applications that exploit the IoT, AI, and cloud computing to provide rich and personalized customer experiences. Such applications are usually developed in an ad-hoc fashion, typically as technology showcases, often with little validation in the field. Indeed, there is no methodology to elicit and specify the requirements for such applications, nor domain level reusable components that can be leveraged to implement such applications with the context of e-commerce solutions. An e-commerce software vendor asked us to do just that, in a domain with a fragmented scientific literature, and with no portfolio of applications to draw upon. In this paper, we describe our domain engineering *strategy*, present the main elements of the *technical approach*, and discuss the main difficulties we faced in this domain engineering effort. Our approach is intended for marketing analysts and customer experience designers. It offers to them a methodology and tools to design customer experiences and generate building blocks of CXM functionalities to be integrated in e-commerce suites of their customers—the retailers.

Keywords Domain engineering · Customer experience management · Cognitive modeling · Ontologies · Metamodeling · Metaprogramming

1 Introduction

Jane walks into her favorite grocery. As she drops items in her shopping cart, the food labels are displayed on her phone or a head-up display. As she drops a box of crackers, she is warned of the sodium level, given her blood pressure. Walking through the produce section, she gets notices about the latest arrivals of fair trade certified products, being an active member of an environmental advocacy organization. Walking into the meat section, an MMS mentions a special on lamb chops that her significant other enjoys. As she drops a rack into her cart, she receives two thumbs up for a Syrah wine from northern Rhone, and one thumb up for a Merlot. While getting toothpaste, she gets an SMS about the special on size 4 diapers, since she has been buying size 3 diapers for the past six months!

Six years ago, the CEO of an e-commerce software vendor presented us with this scenario and asked "what software frameworks do I need to include with my product so that my customers [retailers] can design [i.e. *specify*] and integrate [develop and integrate] such experiences into their implementations of our solution". Customer Experience Management (CXM) denotes a set of practices, processes, and tools that aim at personalizing customer's interactions with a company around the customer's needs and desires [30]. This personalization depends on the type of service or product, the type of customers, and how much a company knows about them. Effective personalization depends on communicating the right information (*what*), in the right format

✉ Imen Benzarti
benzarti.imen@courrier.uqam.ca
http://www.latece.uqam.ca

Hafedh Mili
mili.hafedh@uqam.ca

Renata Medeiros de Carvalho
R.Medeiros.de.Carvalho@tue.nl

Abderrahmane Leshob
leshob.abderrahmane@uqam.ca

1 LATECE Lab, Université du Québec à Montréal, Montreal, Canada

2 Eindhoven University of Technology, Eindhoven, The Netherlands

3 ESG-UQAM, Université du Québec à Montréal, Montreal, Canada

(*how*), at the appropriate time (*when*); otherwise, customer *experience* becomes customer *harassment*. In software engineering terms, we were tasked to develop a *domain-specific framework* to help retailers *instrument* existing purchasing processes as they are managed by their e-commerce solutions. This *instrumentation* may involve enhancing the information communicated to the customer—the food label display, and the various warnings in the scenario above. It could also *augment* the process, by *adding extra steps* not typically covered by the e-commerce solution—e.g. the lamb chops and diaper suggestions in the scenario above.

To solve this problem, we need to perform four tasks, in sequence. *First*, we need to understand the *purchasing process* (or *customer journey*), modulo *variabilities* that account for the different *types* of products; for example a detergent versus a car. This understanding needs to be captured in a model that is independent of an IT implementation, i.e. some sort of a *Computation Independent Model* of purchasing. *Second*, based on this understanding, we need to identify when it is *possible* and *appropriate* to 'enhance' this process, and *how*, based on business requirements (more 'purchases', healthier purchases, brand loyalty, etc.). *Then (third)*, we can start specifying the *user* and *functional requirements* for such enhancements, and *finally (fourth)*, develop reusable artifacts that enable us to implement such enhancements.

The *technology* for implementing such scenarios is available: the IoT, for context-awareness, machine learning *libraries*, for various analytical capabilities, and cloud computing, for a distributed virtual computing infrastructure. Software engineering work on *context-aware* IoT-enabled applications does address some of the computational and architectural issues that underlie such applications, including event loops, integration, middleware, cloud - edge load distribution, etc. [1,14,25,26]. Such work addresses some of the issues in the last two tasks above, but provides little help with the first two. Accordingly, we turned to the marketing literature, the social psychology of *consumer behavior* (e.g. [4,5]), and the *service design* literature (e.g. [13,33]), to help fill-in *some* of the gaps. Then, we faced the challenge of translating this understanding into software *requirements*, and, *eventually*, reusable software *artifacts* [23].

Section 2 presents the principles underlying our approach. We start by presenting a *cognitive model* of the purchasing process (Sect. 2.2), before we show steps to operationalize this model in software requirements for a CXM application (Sects. 2.3–2.5). The resulting software framework is summarized in Sect. 3, and the different pieces presented in Sects. 4, 5, and 6. The first tool in the chain is a *requirements elicitation tool* destined to *marketing analysts* to help them specify the *user requirements* for a *purchase scenario*. It is presented in Sect. 4, and consists of two components, one for eliciting the *process/control* aspects of a purchase scenario (Sect. 4.1), and one for eliciting the data aspects (Sect. 4.2). The next tool in the chain, described in Sect. 5, takes the requirements produced by the first tool and generates technology-independent software specifications (*platform independent model*, or PIM). The final tool, described in Sect. 6, generates CXM code in a target technology (a *platform-specific model*, or PSM), which is Java in our case. We discuss the challenges raised by this domain engineering effort in Sect.7, and conclude in Sect. 8.

This paper is a significantly extended version of [8], in terms of new contributions and length. We revised *significantly* the representational metaphor for purchasing processes, which is central to both the *conceptual framework* (Sect. 2) and the *software framework* (Sects. 4, 5, and 6). Indeed, we used in [8] a *process-driven* (BPMN) metaphor to represent *consumer journeys* through what is essentially a *cognitive process* of a *human actor* reacting rationally and emotionally to new pieces of information as they arrive. That representation leads to processes of unmanageable complexity, because of the many potential execution paths. For this paper, we revised the representation metaphor from a *process-driven* to a *case-driven*, and used the *Case Management Modeling Notation* (CMMN) to describe customer journeys. This simplified both the underlying concepts (Sect. refsec:conceptualspsframeworkspsCXM), and the software implementation (Sects. 3–6), and enabled us to make significant progress in the tool implementation. It also led to a new set of issues, discussed in Sect. 7, thanks to the progress made since [8].

2 A conceptual framework for CXM

2.1 Overview

Customer experience (CX) is the set of cognitive, affective, and sensorial responses customers have to any direct or indirect contact with a company during all stages of the purchase process including pre-purchase, purchase, consumption, and post-purchase [22]. Customer experience *management* (CXM) denotes a set of practices, processes, and tools, that aim at personalizing a customer's experience with a company around the customer's needs and desires [30]. The *majority* of e-commerce sites today include some CXM functionalities. When you add a book to your shopping cart on Amazon, it suggests books that other people bought along with yours (*collaborative filtering*); when you book a flight on Expedia, you are offered a discount on accommodation (*cross-selling*). But these 'experience-enhancing' prompts are not always opportune and are of uneven quality, both in terms of modality and content. Further, they do not come "out of the box" of e-commerce software which offer, at best, machine learning libraries. Going back to the research mandate given to us by our industrial partner: to develop an add-on framework to his e-commerce suite to help build *effective* CXM functionalities in a systematic and cost-effective way.

The first step in building reusable software artifacts for CXM (*domain engineering*) is to identify the *requirements* for such artifacts, before we can translate them into software specifications, designs and code. In essence, consumers are dynamic systems whose processes (stay alive, pursue happiness, etc.) require a number of resources (e.g. food) or states (e.g. fitness), triggering consumption processes to replenish the resources ("we are out of milk") or to attain those states ("I need to exercise"). Commercial enterprises are also dynamic systems that build products to sell to consumers. The 'customer experience' is where the consumer's purchasing process meets the enterprise's selling process. CXM aims to manage the interactions between a consumer and a provider, each executing its own processes and pursuing its own objectives. To properly manage this experience, an enterprise needs to *first*, answer the following questions:

- *Q1* what are the steps of the customer purchasing process? In particular: (1) what are the *decision points* in this process, and (2) which *factors* influence those decisions? Example *decision points* include the very decision to *initiate the purchasing process* for a particular product; I could use the latest iPhone, a bigger car, and more RAM for my laptop, but I decided to go for the RAM. Other decisions include the product specs, the retailer, etc.,
- *Q2* which of these steps *requires* or *lends itself* to interactions between the consumer and the enterprise? Indeed, customization can only happen at touch-points between the two processes: the consumer's and the seller's,
- *Q3* how to customize such interactions to 'enhance' the overall *experience*?

Here, 'enhance' can mean make faster, make more pleasant, provide more relevant information, or spend more at the cash register. In turn, Q3 leads to three sub-questions. First (Q3.1), what kind of customizations to offer? The scenario showed several examples, including different flavors of *recommendation*, based on purchase history (lamb chops, diapers) and product associations (Syrah wine), and *dissuasion*, based on medical history (crackers). Second (Q3.2), what data is needed to support those customizations? As we just saw, purchase history, product assortment, and medical history are helpful, but can knowledge of consumers' mental states and the factors influencing their decisions be used? Third (Q3.3), how to obtain that data, in an ethical and privacy-preserving way?

In the remainder of this section, we will present our strategy for answering the three questions.

2.2 A cognitive modeling of the purchasing process

To answer these questions, we relied, in part, on a *cognitive modeling* of the purchasing process. Consumer behavior has been studied thoroughly by social psychologists trying to understand its mechanics (see e.g. [4–6]). The different theories recognize that purchasing decisions are determined by a combination of objective and rational factors - such as the ability of a product to fulfill a biological need - and subjective or irrational factors such as self-image (e.g. being fashionable), and personal values (e.g. being eco-friendly). Bagozzi [5] proposed a model that integrates all of the influences that have been identified by researchers. The model takes 'consumption' in a broad sense, to account for both the *acquisition of a product or service*, such as buying a computer or subscribing to a video streaming service, and *adopting a behavior*, such as dieting or exercising. The model identifies the different steps and the factors that are known to influence the decisions at those steps [5].

Figure 1 recapitulates the elements of Bagozzi's model. To facilitate the understanding of the model, we represent it within the context of a Business Process Management Notation (BPMN)-like view of the *purchase process*, consisting of ordered *steps*, along with the factors that influence them. We later see in Sect. 4.1.2 that the Case Management Modeling Notation (CMMN) is more appropriate to represent customer journeys.

In the *goal desire* step, the consumer may desire to acquire various products—e.g. purchasing a van to carry the kids to soccer practice or buying a Harley Davidson motorcycle to go cruising during the week-end. Figure 1 shows that the onset of such desires is influenced by *Goal feasibility*, *anticipated positive emotions* from acquiring the product, *anticipated negative emotions* from failing to acquire the product, the probability of succeeding in acquiring the product (*outcome expectancies*), how the customer views herself/himself, e.g. as a family man or soccer mom, versus an adventurous rebel (*social identity*), and how often did the consumer buy a van or motorcycle in the past (*frequency of past behavior*).

The *goal intention* step corresponds to the stage when the consumer settles on one of the desires (buying a van *OR* buying a motorcycle) that then becomes a *goal*. In the *implementation intention* step, s/he makes a plan to achieve the goal, identifying the steps needed to reach it—e.g. look for a dealer and find a financing plan. Once s/he starts the plan, s/he moves to the step of *trying*—e.g. visit a dealer on site. *Trying* involves monitoring progress towards the goal, and adjusting the plan. The *Goal-directed behavior* step refers to the 'final act' in the process: acquiring (*purchase*) or using the product. In *goal attainment/failure* the consumer assesses whether s/he has reached the goal. Based on this assessment, the consumer can adjust any of the choices or actions made

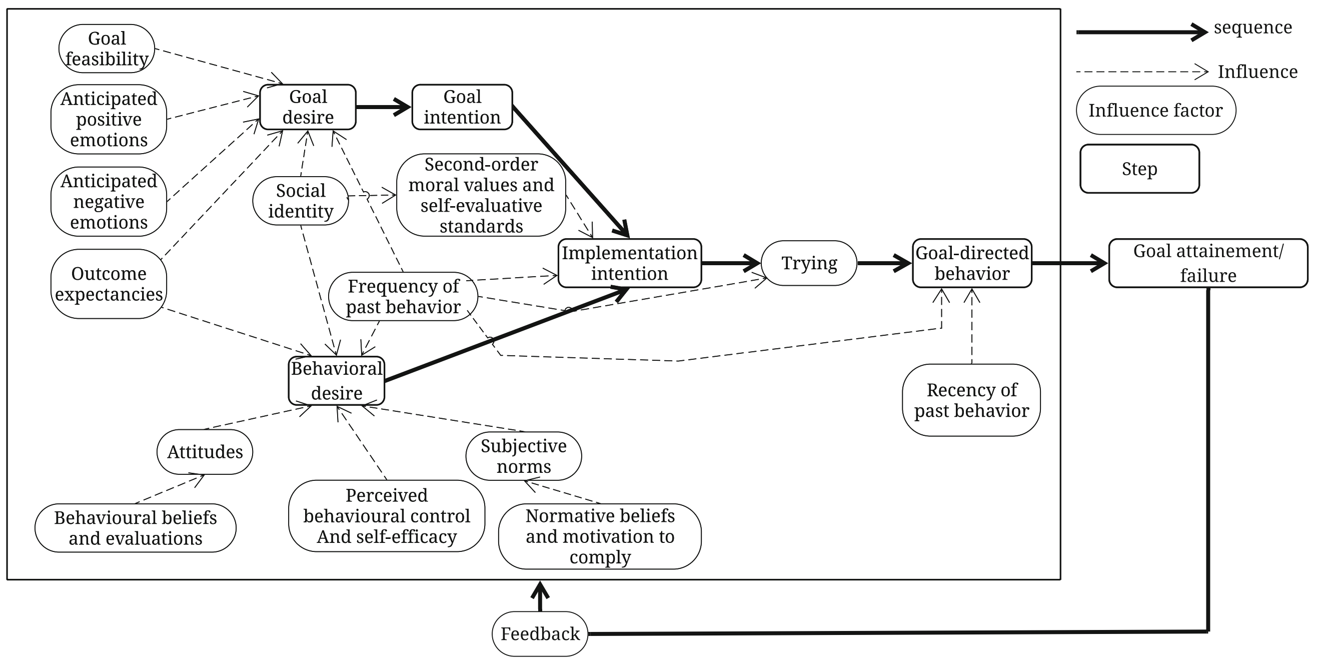

sequence
Influence
Influence factor
Step
Goal feasibility
Anticipated positive emotions
Anticipated negative emotions
Outcome expectancies
Goal desire
Goal intention
Social identity
Second-order moral values and self-evaluative standards
Implementation intention
Trying
Goal-directed behavior
Goal attainement/ failure
Frequency of past behavior
Behavioral desire
Recency of past behavior
Attitudes
Subjective norms
Behavioural beliefs and evaluations
Perceived behavioural control And self-efficacy
Normative beliefs and motivation to comply
Feedback

Fig. 1 A generic purchasing process with the salient influence factors [5]

in the 'purchasing process', including the choice of goals to pursue in the *feedback* step [5,23].

There are different paths through this process, depending on the complexity of the consumption decisions [5,23]. The literature has identified three families of purchasing processes [29]. *Extended processes*, qualified as problematic since the customer faces many obstacles (financial, temporal and spatial) [6], include all the steps shown in Fig. 1; these include processes such as purchasing a car or a house. In *routinized processes*, purchasing decisions are made automatically without conscious control. For example, having the habit of buying coffee on the way to work. By developing this repetitive behavior, the customer minimizes the time and energy spent in the purchasing process. Therefore, *desire*, *implementation* and *trying* steps can be removed from the process. *Limited processes* do not involve significant obstacles, but require minimal cognitive effort; for such processes, the *implementation* step can be removed as explained in Sect. 4.1.3.

2.3 Operationalizing the cognitive model to design customer experiences

Bagozzi's model answers Q1 and parts of Q2, but does not tell us when or whether it is appropriate to interact with the consumer (Q2), and how to customize the interaction (Q3). Further, a retailer typically does not *know* the customer, or which stage of the purchasing process they are at, until they initiate an interaction with the retailer. We address both problems using *metaphors* or *patterns*, explained below.

The first metaphor is embodied in the pattern shown in Fig. 2. It recognizes customer experience (CX) as an interaction between two processes—one of buying and one of selling—around a number of *potential* touch-points, and customer experience *management* (CXM) aims at customizing such interactions in a way that helps the parties achieve their objectives. Basically, a consumer has a number of 'ongoing processes' (living, raising kids, pursuing happiness) that generate a number of needs that, in turn, trigger consumption processes. Consumption processes involve a number of steps (see above), some of which are internal (Step i in Fig. 2), i.e. in the 'consumer's head', while others *require* or lend themselves ("would you like to try out this nicer model?") to interactions between the consumer and the retailer (Steps $i-1$ and $i+1$); these interactions can be initiated by the consumer ("do you have these in size 10.5?") or the retailer ("would you consider this nicer model?"). The steps ($i-1$ and $i+1$) may involve decision making that is influenced by *immutable factors* ("RO (Read-Only) influence factor" in Fig. 2), e.g. shoe size, income, or social identity; or *mutable factors* (RW (Read-Write) influence factor), including the desirability of a product ("they look really good on you!", or *anticipated positive emotions* [5]) or its affordability ("we offer financing", i.e. *Goal feasibility* [5]). Thus, a critical aspect in operationalizing Bagozzi's cognitive model is to recognize the *influence factors* as *data points* about the consumer that the retailer can leverage to customize its product offering, answering parts of question Q3.2; how to obtain them (question Q3.3) is a separate issue (see Sect. 4). This is embodied in the *CXM ontologies* described in Sect. 2.5

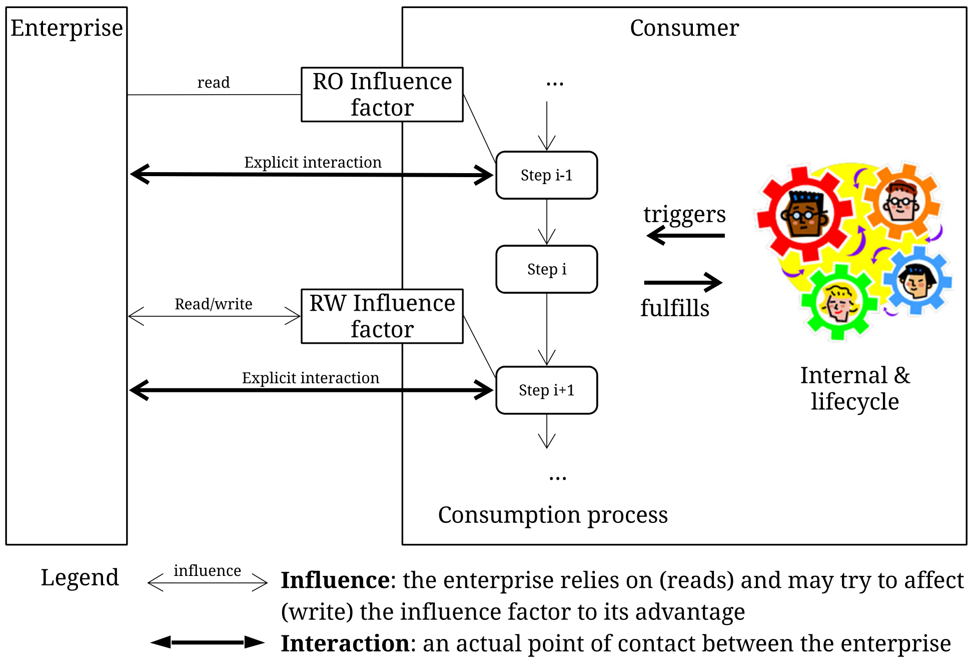


Fig. 2 A pattern for operationalizing touchpoints and influence factors for CXM

This pattern tells only half of the story. As mentioned above, knowing that a step in the *cognitive process* lends itself to an interaction, does not tell us:

1. How to find out whether the consumer is at that particular step at a given moment, especially that different consumers may follow different *journeys* through the purchasing process;
2. Whether it is appropriate to initiate an interaction at this point. Indeed, the fact that a process stage *lends itself* to an interaction does not mean that the retailer should initiate one; we need to ensure that 'customer experience' does not become 'customer harassment';
3. What should the purpose of that interaction be? We can't simply "push" products as the introductory scenario suggests.

A strategy for designing such interactions is presented next.

2.4 Principles for adding CXM interactions

CXM can be thought of as a way to *decorate* the purchasing process in a way that enhances the customer experience. Thus, a convenient way of thinking of CXM functionalities for now, is as add-on functionalities to a no frills e-commerce site that supports straight product catalogue browsing and search, shopping cart management and straight check-out and payment; we will extend this later to a *multi/omnichannel* context (Sect. 4). An add-on CXM interaction can be as simple as the display of an additional piece of information, an *alternate scenario* in a use case involving several user - system interactions and that rejoins the main flow (e.g. adding a site-suggested book to the shopping cart) or a different use case altogether.

To figure out which interactions to add to a vanilla flavor e-commerce site, we rely on principles gathered from the literature in two fields: (1) *psychology*, exploring how individuals interpret sequences of events and their perception of emotionally intense moments, and (2) *service design*, exploring how to design interactions with customers to enhance their service experience. These principles can be summarized as follows:

2.4.1 Interact with customers based on their shopping behavior (P_1)

Recall that a major challenge in designing customer experiences, is to figure out where in the purchasing process the customer is at some point. In [21], the authors identified two behaviors in an e-commerce website, *browsing for information*, and *browsing for pleasure*, and argued that companies should adjust their interactions accordingly[1]. In the first case, a company should wait until the customer initiates an interaction. Indeed, customers have a precise goal and a particular product to buy, and they choose *how* get information, which allows them to adapt the information obtained according to their need and the level of knowledge [2]. By contrast, if customers are browsing for pleasure, the company should initiate two way communication with them and join them

[1] Experienced sales*people* in a brick-and-mortar store are very good at that.

in their hedonic experience (e.g. initiate a discussion with a chatbot).

2.4.2 Assess customers' psychological distance to their Goals (P_2)

Psychological distance measures the distance separating the customer from her goal of acquiring the product [17]. This distance should decrease as the customer progresses in the purchasing process, so that the customer becomes motivated to pursue the goal. Holmqvist identified four types of distances [17]: spatial, temporal, social and hypothetical. This can influence the level of detail of product data to be transmitted to the customer as s/he progress through the process.

2.4.3 Design peaks of intensity level during sequences (P_3)

Peaks, as well as improving trends, enhance the perception of the customer towards the experience; when placed at the end of a sequence, they become more salient and memorable in the long term [2]. In a *surprise peak*, the company presents the customer with an unexpected offer (e.g. a gift, a purchase coupon, etc.).

2.4.4 Partition extended purchasing processes to a set of sequences on strategic moments (P_4)

Extended experiences are usually interrupted—e.g. a customer paused his research of a new car until the end of relocation. With each interruption, customers reset their perception of the experience, and begin a new experience [3]. Indeed, the overall evaluation of the experience is the average of the evaluations of the customer of each of the sequences [2]. Thus, for extended problem solving, we partitioned the process into three sequences. The first sequence contains *desire*, *intention*, and *implementation* steps; the second contains *trying* and *purchasing*; and the third consists of *feedback*.

2.5 CXM ontologies

In this section, we formalize our knowledge about consumers and products in a way that supports the customization of customers' interactions with product/service providers. This knowledge is embodied in *ontologies* in the knowledge representation sense, i.e. as a specification of a set of representational primitives used to represent a domain of knowledge [15]; and in sense of reflecting a *shared conceptualization of a domain*—in this case, consumer behavior.

2.5.1 Consumer data

Knowing the consumer is key to a successful CXM. An enterprise may use information about customers to: (1) anticipate their *needs*, before they even engage in a purchasing process, (2) identify those among its products that best address those needs, consistent with customers' known attitudes, biases, emotions, and values, and (3) present those products in a way that appeals to them. Figure 3 shows excerpts from the consumer ontology. We will comment on the main pieces; full discussion can be found in [23]. The upper half of Fig. 3 shows that *Consumer*'s (individuals, such as Jane) belong to *Category*'ies (e.g. Yuppies, or Young Urban Professionals). Both have *Property*'ies, such as *income* or *age*, and values for those properties, represented by class *PropertyValue* for individuals (Jane has a single age value), and class *ValueRange* for categories (*yuppies* have an age range from 25 to 40). Both individual's membership to categories, and property values, are qualified with a *confidence level*, reflecting *probability* or *strength* [8]. The lower half of the customer ontology embodies the *life-cycle theory* in marketing [13], where individuals go through different *stages* in life, each having different consumption behavior in terms of needs, attitudes, etc. Life-cycle stages are represented by the class *State* (e.g. married with children), with their own properties and property value ranges. *StateTransitions* are triggered by events (*EventType*), which can be property change events (*PropertyChangeEventType*), as in getting married, or timer events (*TimerEventType*), when a state expires after a given time period—the diapers example in the introductory scenario.

2.5.2 Product data

To take full advantage of our knowledge about the consumer, we need a commensurately rich representation of the products and services sold by the company. Previous work focused on modeling products to configure them and manage their manufacturing life-cycle [11,20]. Product modeling is commonly multilayer, where we distinguish the category of the product, model of product and the physical item. Space limitations does not allow us to go into the details. Suffice it to say that our product ontology supports: (1) different levels of instantiation—the multi-layer idea, (2) product assortments (lamb with red wine), (3) includes information about *physical attributes* (e.g. dimensions), *functional attributes* (what they are for), (4) *packaging* (presentation, aesthetics, etc), and (5) *manufacturing process*, to include the kinds of properties that can be matched to consumers' attitudes and *second-order moral values* [5,23].

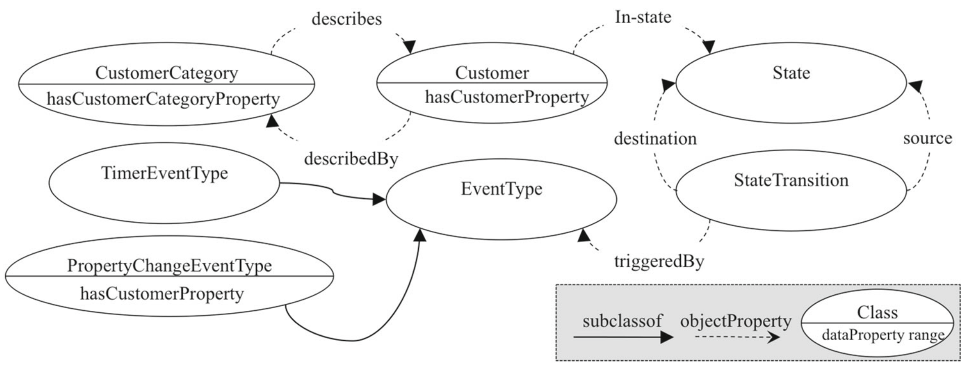


Fig. 3 Excerpts from the customer ontology metamodel

2.5.3 Other data

The full spectrum of CXM functionalities require a wide range of data, including products reviews, product comparisons, advertising material, etc. We proposed in [23] a rich representation of advertising material to illustrate the kind of representation we need, and how it could be used.

3 A software framework for CXM

Section 2 laid the *conceptual* groundwork needed to develop a *software framework for CXM applications*. It showed the wide conceptual gap between the *concepts* emanating from (a fragmented) CXM theory, and the *operationalization* of those concepts in *domain software artifacts*. Model-Driven Engineering (MDE) provides a useful guide to decompose the functionalities of our *software* framework, shown in Fig. 4.

As mentioned in Sect. 2.1, the *purchasing process* depends heavily on the type of product or service being sold, e.g. a carton of milk versus an appliance, computer or a car. Section 2.5 showed *representational primitives* needed to support CXM functionalities, but we still need to specify relevant *data models* for a specific product type. Both need to be specified by a *marketing analyst*, who typically is neither a *social psychology* researcher, or a modeling guru!

Accordingly, the first tool in our framework is one for specifying *purchase scenarios*, to be used by marketing analysts that produces a *domain-level, computation-independent model* (CIM) of the purchase scenario. This tool, described in Sect. 4, relies on: (1) an encoding of Bagozzi's model, and (2) a *library of interactions*, from which the analyst can pick ones that are relevant to the process at hand. The contents and format of these *interactions*, which embody the state of the art in *service design*, will be described in Sect. 4.

The next tool in our MDE chain (Domain/Scenario-Specific Generator) takes the specifications produced by the first tool, and the generic CXM ontology, to produce a *platform independent model* (PIM) consisting of a scenario-specific ontology that represents the data, and an abstract description of the process. The current implementation does *not* generate the process component, but generates the specific ontology. The generator is described in Sect. 5.

Finally, the Java Code Generator takes the output of the Scenario-Specific Generator, and a library of CXM function templates, and generates Java code that implements the data layer of the purchase scenario at hand, and the CXM customization functions (e.g. recommendation) tailored to that data (see Sect. 6).

4 A tool for specifying purchase scenarios

The first tool in our tool chain (see Fig. 4) is a *requirements elicitation tool* destined for *marketing analysts* that elicits the customer experience management (CXM) requirements for the *purchase scenario* at hand. We saw in Sects. 2.2 and 2.3 that there are different paths through Bagozzi's model (Fig. reffig:bagozzispsbasic) depending on the complexity of the purchase/product. Thus, a *purchase scenario* is characterized by the specific subset of the generic Bagozzi process that applies to a given product family, and the corresponding data model needed to support CXM for that process. This raises the issue of what is the appropriate *scope* for a purchase scenario; see Sect. 7.4 for a discussion.

The (purchase) scenario specification tool produces a *domain level model* of the CXM application that is used as an input to our MDE tool chain. This model, which corresponds to a *Computation Independent Model* (CIM) of the CXM application, includes an abstract description of the scenario-specific purchase process, along with its CXM enhancements, and an abstract description of the data for the scenario. We discuss the two functionalities in turn.

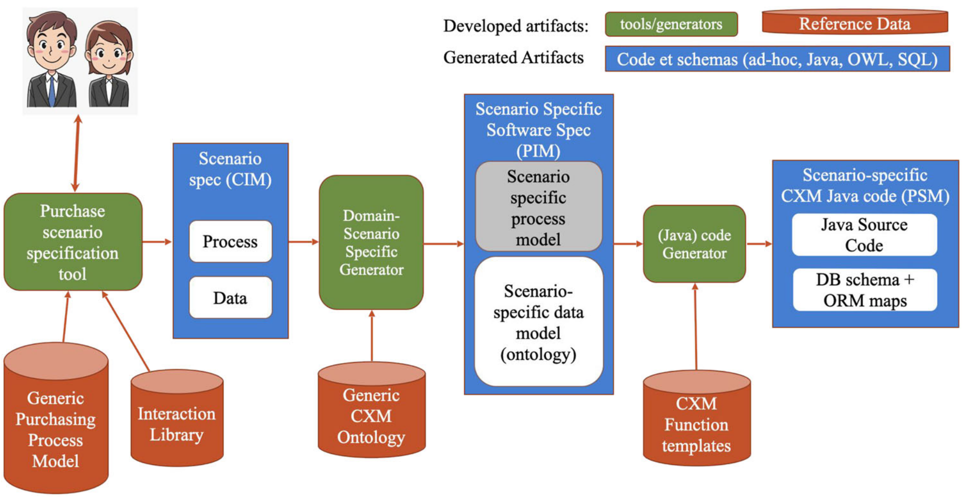

Developed artifacts:
tools/generators
Reference Data
Generated Artifacts
Code et schemas (ad-hoc, Java, OWL, SQL)
Purchase scenario specification tool
Scenario spec (CIM)
Process
Data
Domain-Scenario Specific Generator
Scenario Specific Software Spec (PIM)
Scenario specific process model
Scenario-specific data model (ontology)
(Java) code Generator
Scenario-specific CXM Java code (PSM)
Java Source Code
DB schema + ORM maps
Generic Purchasing Process Model
Interaction Library
Generic CXM Ontology
CXM Function templates

Fig. 4 Software framework for CXM

4.1 Specifying the purchase scenario process

4.1.1 Purchase processes: the case for case management

The purchasing process, as represented in Fig. 1, shows a linear sequence of steps, going from the very onset of the *desire* to acquire a product, to its purchase and use. For all but the most routine purchases, the purchasing process is *seldom* linear. One of the authors responded once to an ad to buy a *spare* helmet for his motorcycle, and ended up buying the seller's motorcycle and *two* helmets. Many purchasing processes start with the intent to purchase product A (*Goal Intention*) end up with the purchase of another product B (e.g. a pricier version of A, or *up-selling*), or A *and* B ("while you are here, would you like to try my motorcycle?"—*cross-selling*), or abandoned, or interrupted to be started over again. All of this happens because in the course of this essentially *cognitive* process, new information comes to light that can change—or skip altogether—any step, be it the (*Goal*) intention, the planning (*implementation intention*), or the plan execution (*Trying*).

Processes that involve mostly human cognitive tasks, that are flexible, and data-driven, are best represented as *cases* [24]. Contrary to business process management (BPM), which is process-driven, embodies routine work, and follows a predefined workflow, *case management* is data-driven, knowledge-worker based, and non-deterministic: the case flow is determined at run-time by the actions of the knowledge worker [28].

For the purposes of our scenario specification tool, we will use the *Case Management Model and Notation* (CMMN) language, which supports the case management requirements [19] and incorporates expertise on case management from many software vendors like Oracle and SAP [27].

In CMMN, the concept of a process corresponds to a *case handling*, and thus, we talk about a *case*. Graphically, a *case model* is enclosed in the picture of a *folder*, as shown in Fig. 5. At a basic level the "handling" of a *case* involves performing *tasks*, and so a *case model* may include different *tasks*, represented graphically by rectangles with rounded corners (*Inventory verification* and *Item delivery* in Fig. 5). In CMMN, a standalone task is, by default, enabled, i.e. it can execute. To make its execution *conditional*, we add an *entry sentry*, which is represented graphically by a hollow diamond on the left of the task. In CMMN, we use *stages* to group together tasks that are activated/enabled together. Sentries apply to stages as well. Graphically, stages are represented by rectangles with cut corners (*Payment* and *Delivery* in Fig. 5). Case models and stages may include *user events*, which correspond to actions undertaken by human actors. Fig. 5 shows *Proceed to payment* as a user event taking place with the *Payment* stage.

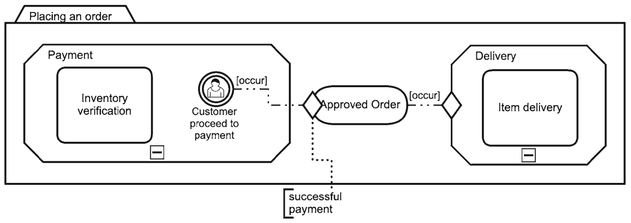


Fig. 5 A CMMN model example: place an order process

Absent an explicit sequence of tasks through which a case progresses, we use *milestones* to indicate such progress, represented graphically by oblong shapes. Figure 5 shows *Approved Order* as a *milestone*, which is connected to the entry sentry for the *Delivery* stage. This means that once we reach that milestone, we enable the *Delivery* stage, hence activating all the tasks within.

In CMMN, a case model with two tasks T_1 and T_2 and nothing else means that we can perform T_1 or T_2, any number of times, in any order, including none at all: whatever we *add* to a model *constrains the set of possible executions of the model*. Therefore, CMMN is declarative/descriptive. This is different from BPMN where the only possible executions are through the specified paths. Even *escalation events*, which were introduced in BPMN 2.0 to handle ad-hoc user events, should trigger an event sub-process for each customer event, in our case, this will make purchasing process steps overlapping. BPMN is an imperative/prescriptive modelling language, *what is not explicitly authorized/specified is forbidden* whereas in CMMN, *what is not explicitly forbidden is possible*. This difference has important implications on the *relation* between *specific executions* of a case model, and *the case model itself*: "case workers" choose their own execution paths, among those *allowed despite the constraints placed in the case model*.

4.1.2 A representation of Bagozzi's process and of the CXM interactions

Our tool embodies a CMMN-based representation of the generic purchasing process, consisting of Bagozzi's cognitive model, shown in Fig. 1, augmented/decorated with interactions designed using the principles outlined in Sect. 2.4. The marketing analyst chooses those among the steps and interactions that are relevant to her/his purchase scenario, as explained in Sect. 4.1.3; in this section, we show the CMMN representation of the generic process.

Using CMMN, process steps in Bagozzi's model (see Fig. 1) are mapped to CMMN *stages*, to which we add the stage *Unknown* (Fig. 6), explained below. *Stages* are containers used to decompose a complex case into 'episodes', to reduce the complexity of the case. The Unknown stage represents the stage in the process, from the point of view of the retailer, when the retailer does not know how far along in the purchasing process the customer is. It is the *initial stage* of a *customer experience*; as soon as the customer initiates an interaction with the retailer, the retailer can make inferences to guess where the customer is, and moves the process to a different stage.

When a stage is reached, all the items contained in it are activated. Per CMMN's semantics, stages are independent. Thus, a marketing analyst can remove stages that are irrelevant to her/his purchase scenario in the configuration phase (Sect. 4.1.3). At *run-time*, an actual customer can start the "process" at any stage whose entry conditions—if any—are satisfied, or dropout out of any stage.

Figure 7 shows how *interactions* connect *stages*. We distinguish between two types of interactions: (1) *tasks*, which represent interactions initiated by the retailer in the current stage to implement a particular marketing strategy (see Sect. 2.4); and (2) *user events*, which represent interactions initiated by the user. Many interactions lead to a *milestone*, which marks the end of a stage and the beginning of another (Fig. 7).

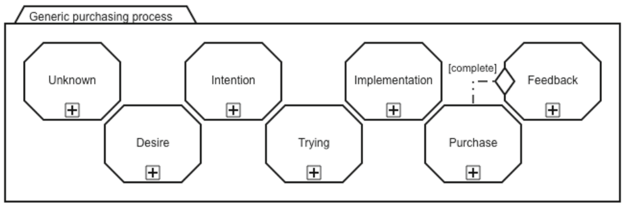


Fig. 6 High level view of purchasing process with CMMN stages representing the process steps

There are two distinct phases: one during design time and other during run time. In design time, the marketing analyst should specify: (1) every possible interaction that might take place with the customer (*tasks*); (2) under which conditions each interaction should happen (constraints using *sentries*, which might be activated by the triggering of *events*); (3) in which context is each interaction (*stage*); and (4) how to evaluate the progress of the case (achievable *milestones*). In run time, the customer is the "knowledge worker" and decides to which interactions he/she will (re)act. Depending on how the customer experiences the progress of the case, he/she determines how the case proceeds.

Figure 8 shows the complete process. This process is generic and independent from the data of the purchasing scenario. It can be customized for a particular purchasing scenario by removing non-necessary steps or interactions and by personalizing interaction properties with scenario data like the communication channel or the exchanged customer and product properties during the interactions. We recall that this generic process is designed according to the principles described in Sect. 2.4, particularly, the assignment of interactions to the process stages. We briefly discuss the process' stages and some of the salient interactions.

Unknown As mentioned above, this is the initial stage of a customer experience (CX), where the retailer does not know where the customer is in her/his process. Most of the user interactions initiated in this stage provide *enough insight* for the retailer to infer the actual stage of the customer in their current purchase process. Thus, many user interactions occurring in the *Unknown* stage lead *out* of the *Unknown* stage and *into* the other stages. In particular, the model shows two *user events* that occur in the *Unknown* stage that embody the P_1 principle, described in Sect. 2.4: distinguish between customer's browsing patterns, be it for *information* or for *pleasure*.

Desire This is the stage where the consumer does not yet known what s/he wants, and it can be inferred from "watching" the consumer's browsing behavior. Hence, one of the interactions leading into this stage is the determination that a consumer, in *Unknown* stage, is browsing for pleasure. A

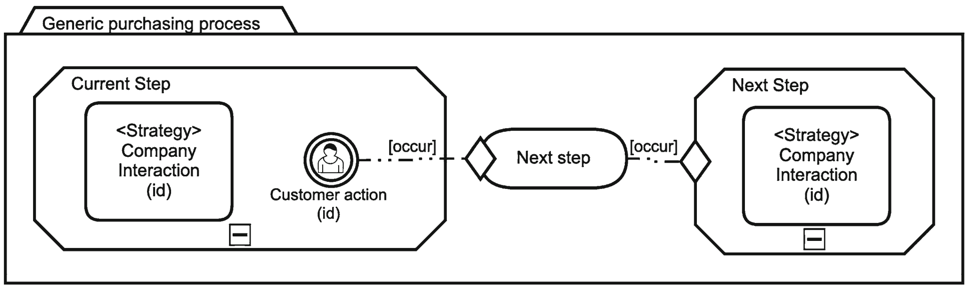

Generic purchasing process
Current Step
<Strategy> Company Interaction (id)
Customer action (id)
[occur]
Next step
[occur]
Next Step
<Strategy> Company Interaction (id)

Fig. 7 Modelling interactions using CMMN

consumer whose behavior changes by showing interest in a specific product (e.g. s/he does a product search on the website, or asks a question about a product to a chat-bot) moves to the *intention* stage (Fig. 8).

Intention In this step, we distinguish between identified (e.g. authenticated) customers and unidentified/unknown ones. Identified customers have a recorded profile that includes data such as preferences and previous purchases, in which case the retailer is able to propose a personalised recommendation based on that profile (see Table 1). If the customer is unidentified, the company can settle for recommending similar products. A customer who responds to these recommendations, for example by opening an email that contains a personalized hyperlink to further information about the recommended product and selecting that link, moves to a next stage of the process (see Fig. 8).

Implementation (intention) Recall that *implementation intention* is the planning stage. For complex purchases, making a purchase plan may be *long* and *difficult*, especially for a customer experiencing the purchase of a product for the first time; this is due to an important *psychological distance*, such as the lack of information about the product or financial resources. As per principle P_4, the retailer can propose a *pause* in the process at this juncture, e.g. by proposing a channel switch (task T7, see Fig. 8 and Table 1). As per principle P_3 (design peaks at the end of a sequence), the retailer can ensure that this sequence (desire $\rightarrow$ intention $\rightarrow$ implementation) concludes with a positive note, e.g. by offering a discount coupon (task T6 in Fig. 8 and Table 1). Figure 8 shows that the tasks that represent these interactions can be activated after a time delay configurable by the analyst (timer event listener in CMMN).

Trying If a customer is blocked in this stage, it means that the psychological distance that separates them from their goal prevents them from taking the action of Purchase. Per principle P_2, interactions in this stage aim to decrease this distance, either by recommending a restricted set of products to guide the customer in his choice; or by communicating with them through expert advisors. Such communication may be initiated by the customer (see Fig. 8 and Table 1).

Purchase For a long purchasing process (e.g. for a car), this is the end of the second sequence (see principle P_4). Per principle P_3, a retailer can propose a peak interaction such as discount coupons and free samples to enhance the customer's affective response (T11 and T12, see Fig. 8 and Table 1). Per P_2, if the product requires delivery, and the retailer anticipates potential delivery delays, it may act proactively by tracking delivery (T16) and notifying the customer (T15)—which has been shown to decrease the perceived duration of delay [10].

Feedback The interactions proposed by the retailer in this stage depend on the customer review (see Fig. 8). Positive reviews may create an opening for cross-selling (T21), upselling (T17), and repeat purchases after some time elapse (T19), whereas a negative review should compel the retailer to follow-up with the customer (T20). And so forth—see Fig. 8 and Table 1.

In the next section, we show how a marketing analyst can personalise this generic process for a specific purchasing scenario.

4.1.3 Configuring a purchase scenario's process and CXM functionalities

We developed a tool that enables marketing analysts to specify the purchasing process, decorated with CXM functionalities/interactions, for the purchase scenario at hand. The final process, with its CXM functionalities (interactions) for a specific scenario, will be a *subset* of the generic process presented in the previous Section (see Fig. 8). To facilitate the specification of such scenarios, we provide the analyst with customisable process templates that they can edit, by adding or removing stages, system tasks (see Table 1), or user events (see Fig. 8). For the purposes of the prototypes, we provide three such preconfigured processes corresponding to a *complex process* (the generic process in full), *limited processes*, and *routinized processes*, e.g. for buying a carton of milk or diapers.

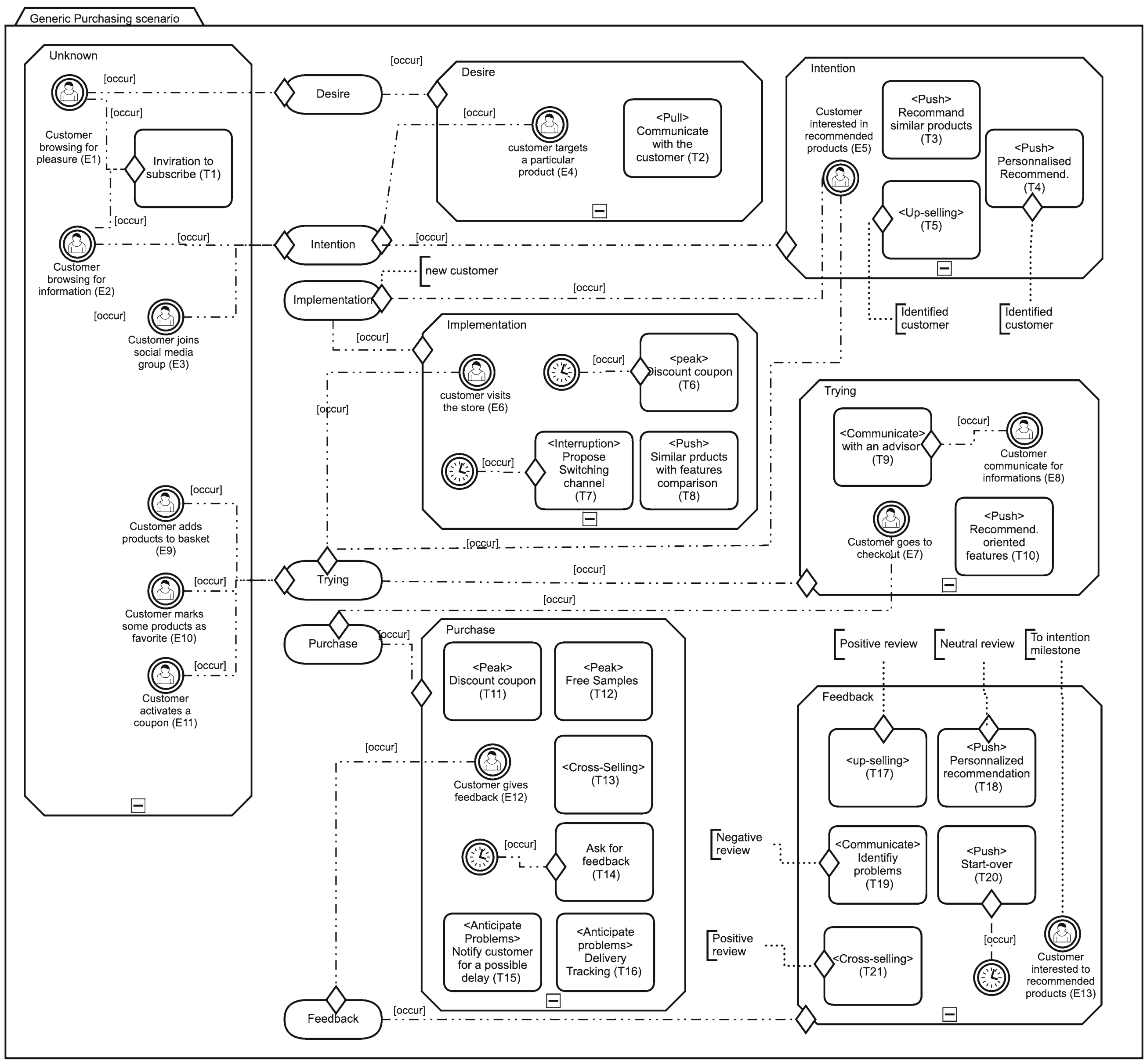

Generic Purchasing scenario
Unknown
[occur]
Customer browsing for pleasure (E1)
Invitation to subscribe (T1)
Customer browsing for information (E2)
Customer joins social media group (E3)
Customer adds products to basket (E9)
Customer marks some products as favorite (E10)
Customer activates a coupon (E11)
Desire
Intention
Implementation
Trying
Purchase
Feedback
new customer
customer targets a particular product (E4)
<Pull> Communicate with the customer (T2)
Customer interested in recommended products (E5)
<Push> Recommand similar products (T3)
<Push> Personnalised Recommend. (T4)
<Up-selling> (T5)
Identified customer
customer visits the store (E6)
<peak> Discount coupon (T6)
<Interruption> Propose Switching channel (T7)
<Push> Similar prducts with features comparison (T8)
<Communicate> with an advisor (T9)
Customer communicate for informations (E8)
Customer goes to checkout (E7)
<Push> Recommend. oriented features (T10)
<Peak> Discount coupon (T11)
<Peak> Free Samples (T12)
Customer gives feedback (E12)
<Cross-Selling> (T13)
Ask for feedback (T14)
<Anticipate Problems> Notify customer for a possible delay (T15)
<Anticipate problems> Delivery Tracking (T16)
Positive review
Neutral review
To intention milestone
<up-selling> (T17)
<Push> Personnalized recommendation (T18)
Negative review
<Communicate> Identifiy problems (T19)
<Push> Start-over (T20)
<Cross-selling> (T21)
Customer interested to recommended products (E13)

Fig. 8 CMMN model of the generic process based on Bagozzi's model (Fig. 1)

Figure 9 shows a screenshot of the current interface of the tool. It contains an editor pane (center) and a palette (left side) with three segments for, (1) process templates (top-left), (2) various kinds of system tasks/interactions (middle-left), and (3) user actions (bottom-left). The marketing analyst can drag and drop elements from the palette to the editor pane. The editor pane offers contextual menus for each element to specify other properties like the channel, conditions (entry sentries in CMMN) and exchanged data. The tool is being developed using *mxGraph*, a fully client-side JavaScript diagramming library that uses SVG and HTML for rendering[2], and that serializes the diagrams in an XML format. The tool uses a simplified diagramming notation, appropriate for a marketing analyst who may not be (probably is not) versed in CMMN.

The editor pane in Fig. 9 shows the specification of baby diapers purchasing process. This is a simple, routinized and periodic purchase. The process template for routinized processes omits the stages *Desire*, *Implementation (intention)* (planning) and *Trying* (executing the plan). Here, we go from *Intention* (we are out of diapers) to *Purchase*. In the *Unknown* stage, customers *visiting the website for information* are invited to *log-in or sign-up* to get their historical data if applicable; from there they move to the *Intention*

2 https://github.com/jgraph/mxgraph.

stage. In the *Intention* stage, the company can *recommend similar products*. If the customer *goes to check-out*, the process moves to *Purchase* stage. The reader can check that the tasks and user actions proposed in the *Purchase* and *Feedback* stages are a subset of those included in the full generic process (Fig. 8).

Note that the model produced by the process editor (Fig. 9) is serialized in an XML format that is specific to the *mxGraph* library; it represents a graph where edges and vertices can have an arbitrary number of metadata. At this stage of the specification process, this model is sufficient because: (1) the data model is *yet* to be specified (see next) and thus we cannot be more precise about the data contents of the various tasks and interactions, (2) it has the right level of formality for the average marketing analysts. The next tool in the MDE chain (see Sect. 5) will pull the two pieces of information together.

4.2 Specifying data model

The specification of the data model involves: (1) specifying the *product/service* being offered, i.e. product categories and models, and their properties (see Fig. 10), (2) specifying the customer and customer categories, by selecting among a set of prototypical properties for both, and 3) specifying properties to be added to both *customer* and *product models* to *properly handle the relevant influence factors for the purchase scenario at hand*. We discuss the three aspects in turn.

With regard to products, we use three levels of abstraction: (1) product *categories*, (2) product *models*, and (3) product *items*. For a car, product *categories* include things such as SUVs, Japanese cars, Mazda cars, or Ford SUVs. Product categories can be *nested*. For example, *MazdaSUV* is a subcategory of *MazdaCar and SUV*. A product *model* (e.g. the *Ford Explorer XLT 2020*) belongs to a category. A product *item* is an *instance* of a *model*, e.g. Jane's Ford Explorer. Each one of these levels has its own properties. Category nesting leads to *restrictions* on *properties*. For example, whereas the *manufacturer* of a *JapaneseCar* is the *PropertyValueRange* {Honda, Mazda, Mitsubishi,...}, the *manufacturer* of a *MazdaCar* is the singleton {Mazda}. Figure 10 shows the data specification GUI.

The specification of customer profiles is slightly different: analysts are provided with a predefined list of attributes that are *typically* used in customer profiles to choose from. Analysts have the option to specify *customer categories* with property *value ranges*, to encode preexisting marketing knowledge about consumer behavior. This was a feature requested by our industrial partner, to allow his customers (retailers) to bootstrap the system with "manual customer categories until they gather enough consumer data to create their own", but derided by marketing academics 'sociodemographics are dead; long live big data'.

Table 1 Details about some interactions initiated by company

Interaction	Current stage	Goal	Data
Communicate with the customer (T2)	Desire	Convey information	General information about products and brand
Recommend similar products (T3)	Intention	Decrease psychological distance	Similar products with more/less features
Up-selling (T5)	Intention/feedback	Increase customer value	Higher end products
Personalized Recommendation (T3)	Intention	Decrease psychological distance	Recommendation according to traces of customer's research
Recommend similar products with features comparison (T8)	Implement.	Decrease psychological distance	Similar products Features comparison
Propose switching channel (store to online/online to store) (T7)	Implement.	Enhance customer response	Nearest store Web site address
Discount coupon(T6)	Implement./purchase	Enhance customer response	Discount amount Validity period
Recommendation oriented features (T10)	Trying	Decrease psychological distance	Customer's preferred features. Small set of products
Communicate with advisor (T9)	Trying	Decrease psychological distance	Customer's preferred features
Cross-selling (T13)	Purchase/feedback	Increase customer value	Related products
Notify customers for possible delivery delays (T15)	Purchase	Enhance customer response	A new delivery date

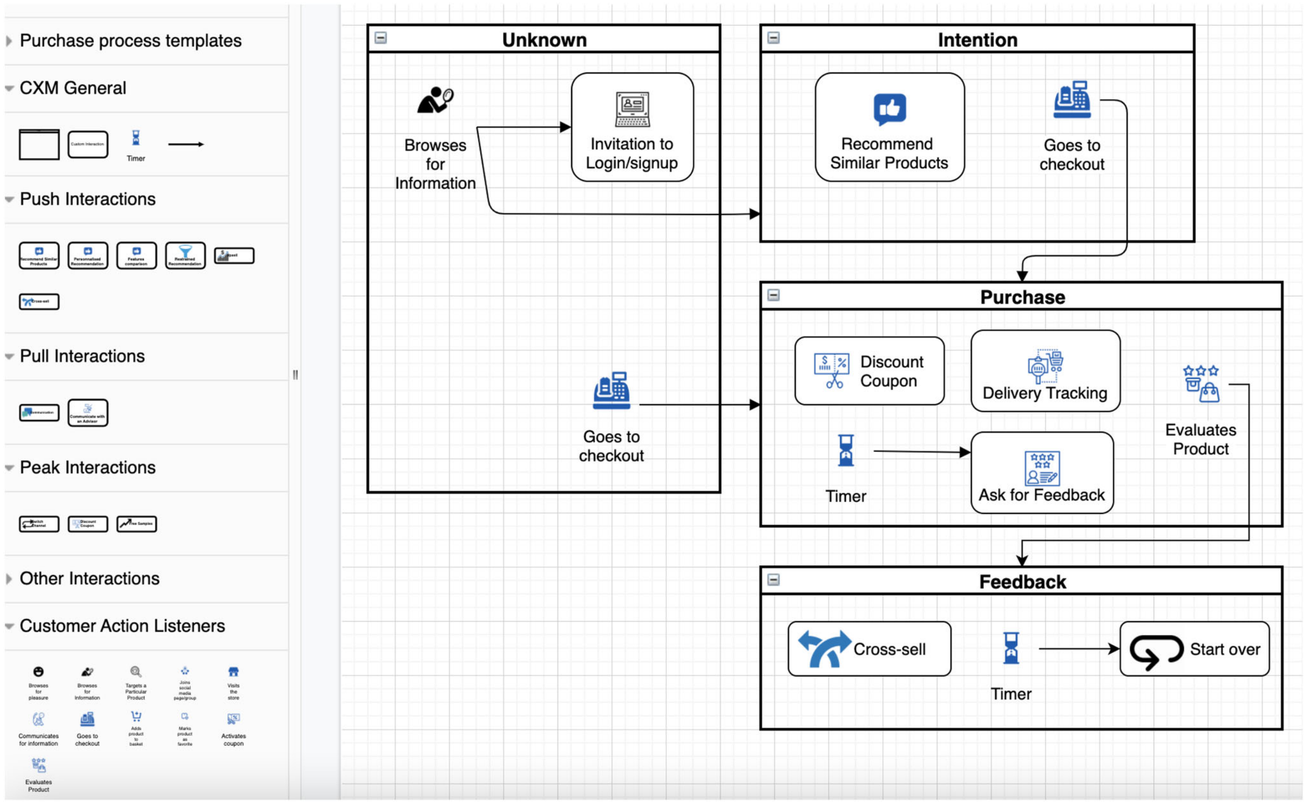


Fig. 9 The GUI for specifying a purchasing process: the example of a diapers purchasing process

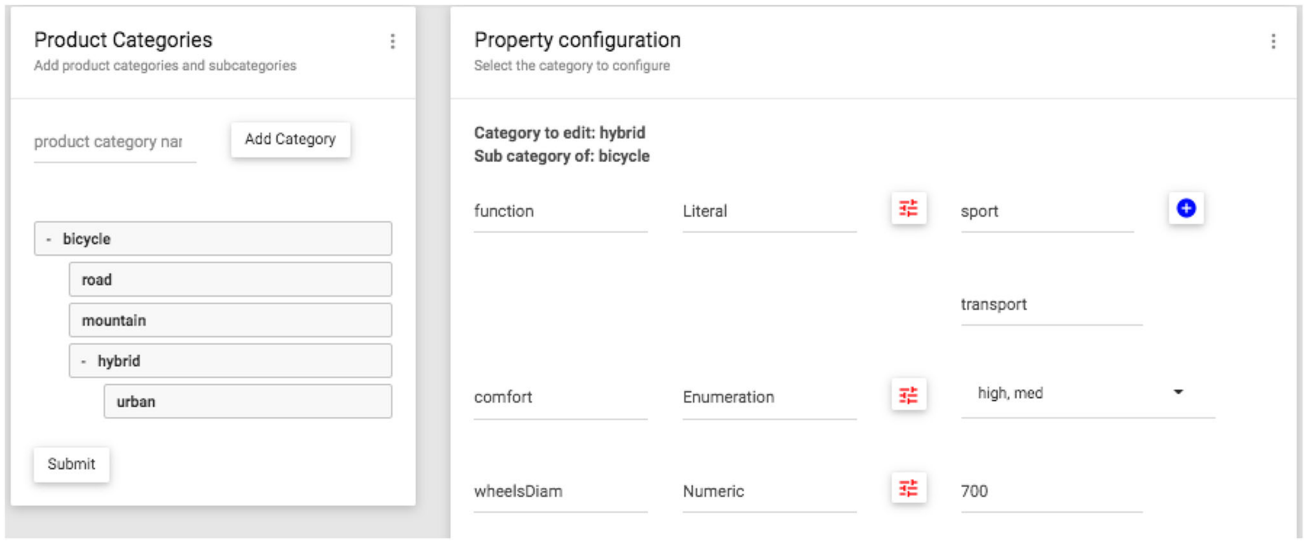
Finally, the analyst is prompted for the properties that are needed, for both products and consumers, to account for the influence factors. Take the example of *goal feasibility*: we know from marketing theory that goal feasibility influences the desirability of a goal (*Goal Desire*), and its likelihood to be retained/selected (*Goal Intention*) as *the object* of the purchasing process (see Fig. 1). The challenge is to identify those characteristics of the product that are potential obstacles to its acquisition, and to record the corresponding characteristic of customer that can help assess their *capacity to overcome those obstacles*.

Take the example of *price*. If price can be an issue for this type of purchase, it helps to record the customer's *income*. Price is not the only obstacle: *yuppies* may be able to afford a fancy treadmill, but *space* might be an *obstacle* if they live in a condo. Thus, we need to make sure that: 1) the product dimensions are recorded, and 2) the "living space" of the consumer is recorded, somehow.

We have encoded 'proof of concept' questions for the *goal feasibility* influence factor in the current implementation of the tool. But as the above example shows, we must carefully analyze each influence factor to figure out: (1) what it could mean in the context of a particular product type, and (2) how to guide the analyst to add the required properties to both product and customer.

5 Generating scenario-specific software specifications

Recall from Fig. 4 that the scenario-specific generator takes the specification produced by the scenario specification tool (Sect. 4), and the generic CXM ontology to produce a platform independent model (PIM). That model (PIM) consists of a scenario-specific ontology and CMMN description of the process. The specification produced by the scenario specification tool (Sect. 4) consists of two XML files representing: (1) the data schema, as elicited by the tool for specifying the data model (Sect. 4.2), and (2) the serialized format of the purchase process specified by the process editor described in Sect. 4.1.3. For all practical purposes, the scenario-specific generator (Fig. 11) can be thought of as consisting of two subsystems, one for generating the scenario specific ontology, described next (Sect. 5.1), and one for generating the CMMN model for the purchase scenario process, described in Sect. 5.2. At the time of this writing, the CMMN model generator is not yet complete, and hence, its output (scenario specific CMMN model in Fig. 11) is greyed out.



Product Categories
Add product categories and subcategories
product category nar
Add Category
- bicycle
road
mountain
- hybrid
urban
Submit

Property configuration
Select the category to configure
Category to edit: hybrid
Sub category of: bicycle

function	Literal	sport
		transport
comfort	Enumeration	high, med
wheelsDiam	Numeric	700

Fig. 10 The GUI for specifying product categories

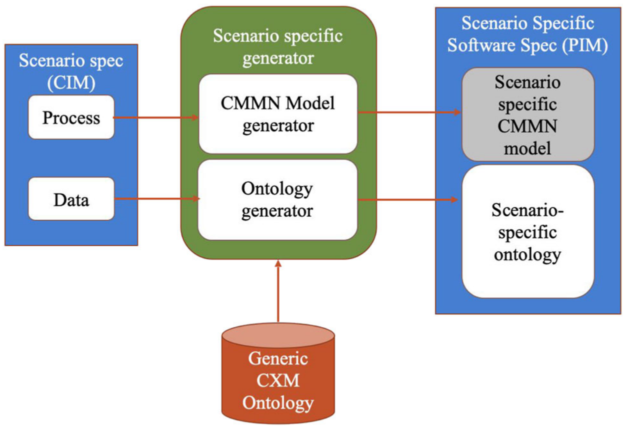

Scenario spec (CIM)
Process
Data
Scenario specific generator
CMMN Model generator
Ontology generator
Scenario Specific Software Spec (PIM)
Scenario specific CMMN model
Scenario-specific ontology
Generic CXM Ontology

Fig. 11 The CIM to PIM scenario specific generator

5.1 Scenario-specific ontology generator

First, note that the generic CXM ontology, excerpted in Fig. 3 (the customer metamodel part), was implemented in OWL using the *Protege* tool (see [7]). Further, the data specification produced by the Scenario Specification Tool (Sect. 4) contains: (1) the product category hierarchy (see Fig. 10), along with the definition of the category properties, (2) product models, along their properties and associated categories, and (3) consumer categories, with their properties, and (4) the consumer profile, with its properties. This information is provided in JSon format, through a REST API of the scenario specification tool.

The scenario-specific generator takes the scenario data specification file, parses it, and generates OWL API code to create, programmatically, the corresponding OWL entities, using the representation primitives of the generic CXM ontology. The generation of the scenario-specific ontology relies on the following rules:

- Each product (customer) category is mapped to an *instance* of the generic CXM ontology *concept ProductCategory* (*CustomerCategory*)
- Each product (customer) category property is mapped to an *instance* of the generic CXM ontology concept *ProductProperty* (*CustomerProperty*)
- If a property P_A of a category A is defined by the analyst as a *restriction* of a property P_B of a super category B, then P_A is defined as a subconcept of P_B, and the restriction is enforced at the range (value type) level.

Figure 12 illustrates the above rules. In this case, the marketing analyst identified the category *Bicycle*, and its subcategory *MountainBicycle*. Both are created as *instances* of *ProductCategory*. Because the analyst stated that *MountainBicycle* ‘restricts’ the *WheelDiameter* property of *Bicycle*, the generator created a separate property for *MountainBicycle* (*MountainWheelDiameter*) that is a subconcept of *WheelDiameter*.

5.2 Scenario-specific purchase model generator

As mentioned in Sect. 4.1.3, we developed a tool that enables marketing analysts to specify the purchasing process. The tool is being developed using the library *mxGraph*. The resulting diagram is serialized in an XML format specific to *mxGraph*. The scenario-specific generator transforms the XML diagram to a CMMN description of the process. Fig. 13 shows the CMMN model that would result from the process

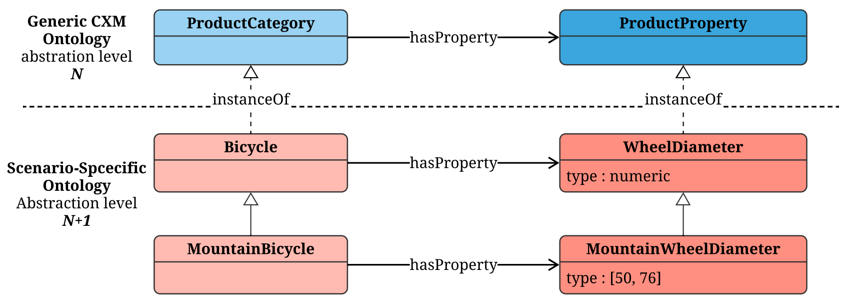


Fig. 12 An illustration of the ontology generation rules for the bicycle scenario [7]

shown in the process editor of Fig. 9 using specification tool to a CMMN process model.

A Diagram in *mxGraph* is expressed in the form of a graph that consists of *vertices* (nodes), and *edges* connecting the nodes. These elements of a graph are called *cells*. A cell object is characterized by an *id*, a parent cell—e.g. an interaction has as parent the corresponding stage, a position (x,y) and a *User Object*. A user object stores the business logic associated with a visual cell. In our context, the business logic is specified by the marketing analyst in the contextual menu of each interaction, and consists on: the *type* of the cell (step, company-initiated interaction, customer action, edge), the cell label, customer interaction channel, conditions and data imported from the data model specification.

Based on the business logic stored in the *user objects*, the CMMN model is generated according to the following transformation rules:

- *Step cells* are transformed to *CMMN stages*,
- *Company-initiated interaction cells* are transformed to *CMMN tasks*,
- *The customer actions* are transformed to *CMMN User events*,
- *Cell labels* constitute the *labels of CMMN elements*,
- *Conditions* are transformed to *CMMN entry sentries*,
- *Channel* and *Data* are transformed to case file that will support the execution of CMMN task by the CMMN engine.

In sum, PIM is an intermediate format of the scenario specification between the business specification by the marketing analyst (CIM) and the low level code source generation at the end of the tool channel (PSM). In the PIM specification, we find the data model (scenario-specific ontology) of the scenario and the process view of the specified customer experience in a computational format. These elements need to be transformed to a low-level source code (PSM): 1) to transform the scenario-specific ontology to a data schema and 2) implement the CMMN process tasks in the form of CXM functionality and generated their source code, so that the process can be executed.

6 Generating technology-specific CXM code

Recall from Figure 4 that the Java code generator takes two inputs: (1) an *abstract software specification* of the CX scenario at hand (PIM), produced by the scenario-specific generator (see Section 5), and (2) a library of CXM function templates. It produces as *output* executable code in the target platform; in this case Java with a relational database for object persistence. As mentioned earlier, the interaction selection functionality (Sect. 4) is yet to be completed, and hence all the downstream functionalities are yet to be implemented. Thus, the scenario-specific software specification (PIM) contains *only* the data model. This model is used for two purposes. First, we use it to generate the Java classes and corresponding databases tables that represent the entities of the purchase scenario (products and consumers). Second, we use it to 'instantiate'/generate the *customization functions* for the purchase scenario at hand. We explain them in turn.

First, using the scenario-specific OWL ontology as an input, we used OWL API 5.1.0, a reference API for creating and manipulating OWL ontologies[3], and Hermit Reasoner (1.3.8), to:

- generate database schemas and fill out the tables to store: (1) product information, including the hierarchy of categories, models, and the corresponding property values, and (2) customer categories, states, and individuals;
- generate the corresponding Java classes, with their attributes and accessors, enforcing property-value restrictions expressed in OWL (see Sect. 5);
- generate the Object Relational Mapping between the Java classes and the relational data store.

More details, including an example, are presented in [7]. With regard to the *customization functions*, illustrated with various types of *recommendations* in the introductory scenario, we sought a way of encoding those functions in a way that does not refer *explicitly* to the data model at hand. We will illustrate the problem with an example.

[3] http://owlapi.sourceforge.org.

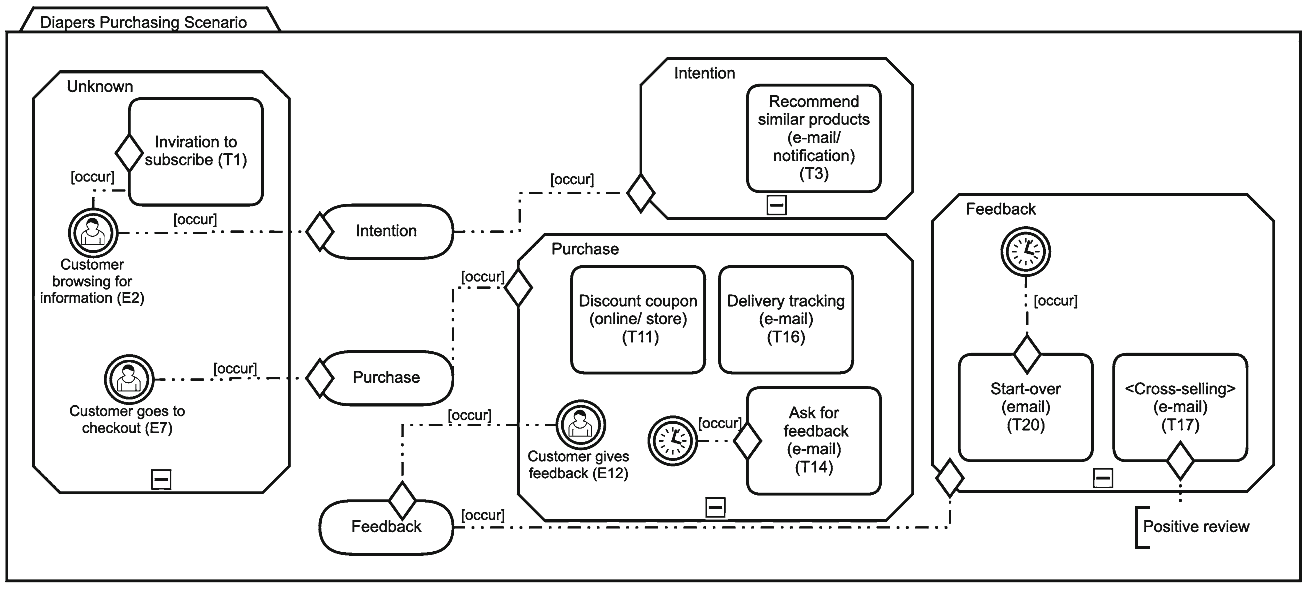

Diapers Purchasing Scenario
Unknown
Inviration to subscribe (T1)
[occur]
[occur]
Customer browsing for information (E2)
[occur]
Customer goes to checkout (E7)
Intention
Purchase
Feedback
[occur]
[occur]
[occur]
Intention
Recommend similar products (e-mail/ notification) (T3)
Purchase
Discount coupon (online/ store) (T11)
Delivery tracking (e-mail) (T16)
Customer gives feedback (E12)
[occur]
Ask for feedback (e-mail) (T14)
Feedback
[occur]
Start-over (email) (T20)
<Cross-selling> (e-mail) (T17)
[occur]
Positive review

Fig. 13 The specification of the diapers purchase scenario in CMMN

Assume that you want to buy a bicycle to ride in the city, e.g. to commute to work. Thus, 'ride in the city' would be the value of a *bicycleUsage* property of customers. Assume now that we have three bicycle categories, *MountainBicycle*, *HybridBicycle*, and *RacingBicycle*, each with a *function* attribute, representing the potential *usages* of the category. Our application should recommend a *bicycle category* bc, if all the *usages* intended by the customer are supported by the category, i.e. if:

$$(\forall\, bc \in BC)\ \text{bicycleUsage(myCustomer)} \subseteq \text{function(bc)} \Rightarrow \text{recommend } bc \text{ models to myCustomer} \quad (1)$$

This encoding refers *explicitly* to *data elements* of the purchase scenario at hand. Further, the matched attributes (*[Customer.]bicycleUsage* and *[BicycleCategory.] function*) do not even have the same name, though they have *similar semantics* and the same *range*.

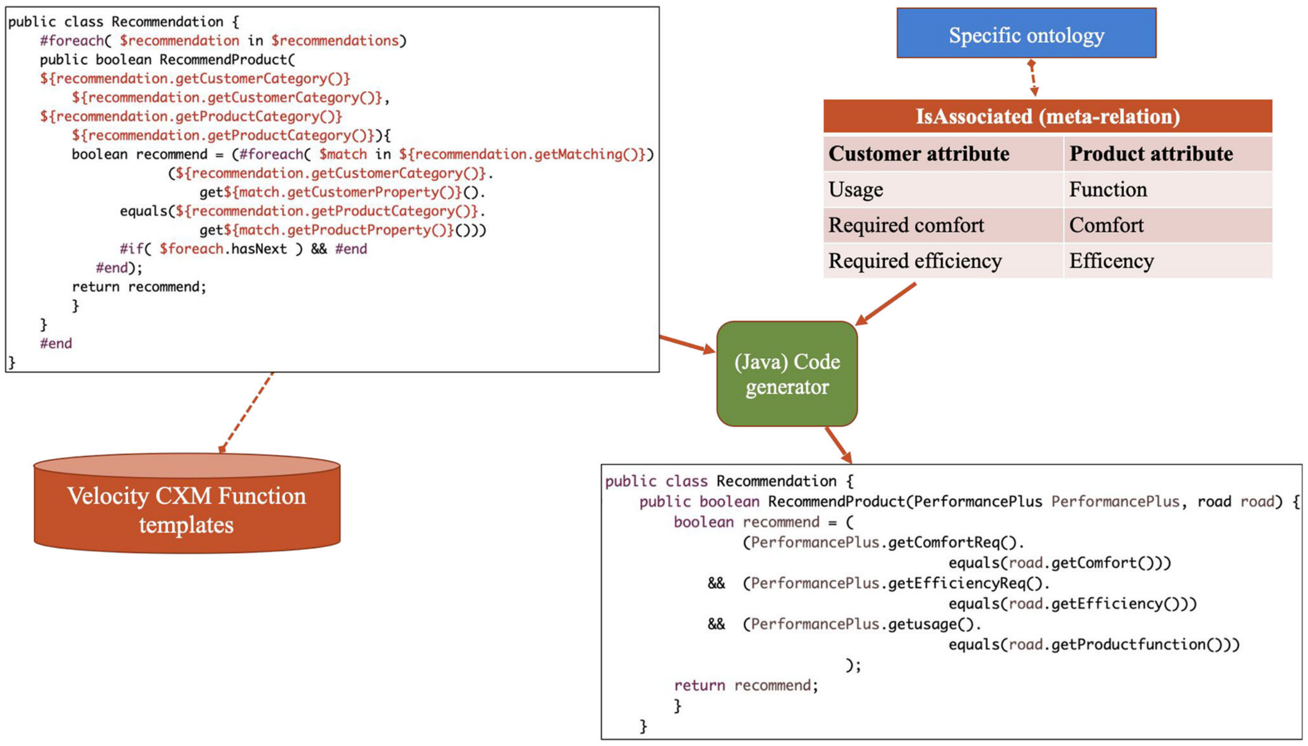
To handle this problem, we sought a codification of CXM customization functions that abstracts away *domain specificity* and focuses on the *intentional* and *computational aspects*. This is like *domain genericity* where the *domain variable* is *semantically constrained*. Thus, we developed a *template language* that relies on a Java-like syntax, and that refers to meta-level constraints on the *scenario-specific ontology* to encode customization algorithms in a domain-independent way (see Fig. 14).

Concretely, we used the *Velocity* templating engine. So far, we encoded a handful of *faceted concept matching functions*, including simple recommendation functions, as proofs of concept. We are currently working on more complex *processing chains*, of the kind we use in machine learning tasks.

7 Discussion

This domain engineering effort was challenging in many respects. Some of these challenges were atypical. First and foremost, customer experience management (CXM) is a novel and immature application domain, which invalidates typical *domain analysis* techniques; this is discussed in Sect. 7.1. Second, we had an unusually *broad scope*, where the domain is neither *purely technical* nor *vertical/industry-specific*. This rendered traditional domain *design/implementation parameterization techniques* difficult to apply. In particular, Sect. 7.2 discusses the difficulties we faced in applying inversion of control—the so-called *Hollywood Principle*, which is key to reuse in application frameworks—to implement CXM-wide functionalities. An important special case of this problem arises for machine learning functionalities, and is discussed separately (Sect. 7.3).

We also faced challenges that are typical of domain engineering. We mentioned in the introduction to Sect. 4 the issue of one versus several purchase scenarios for a given retailer; we discuss this further in Sect. 7.4. Finally, validation is notoriously difficult for *method engineering* research [18], including that involving the *development of domain application frameworks*. We discuss such issues as they pertain to CxDev in Sect. 7.5.



```
public class Recommendation {
    #foreach( $recommendation in $recommendations)
    public boolean RecommendProduct(
    ${recommendation.getCustomerCategory()}
        ${recommendation.getCustomerCategory()},
    ${recommendation.getProductCategory()}
        ${recommendation.getProductCategory()}){
        boolean recommend = (#foreach( $match in ${recommendation.getMatching()})
                    (${recommendation.getCustomerCategory()}.
                        get${match.getCustomerProperty()}().
              equals(${recommendation.getProductCategory()}.
                        get${match.getProductProperty()}()))
              #if( $foreach.hasNext ) && #end
           #end);
        return recommend;
        }
    }
    #end
}
```

IsAssociated (meta-relation)	
Customer attribute	**Product attribute**
Usage	Function
Required comfort	Comfort
Required efficiency	Efficency

```
public class Recommendation {
    public boolean RecommendProduct(PerformancePlus PerformancePlus, road road) {
        boolean recommend = (
                (PerformancePlus.getComfortReq().
                                    equals(road.getComfort()))
            &&  (PerformancePlus.getEfficiencyReq().
                                    equals(road.getEfficiency()))
            &&  (PerformancePlus.getusage().
                                    equals(road.getProductfunction()))
                            );
        return recommend;
        }
    }
```

Fig. 14 Generating CXM Customization Functions

7.1 Why was this hard? CXM: a to be-defined domain

This *domain engineering* effort was particularly *hard*. Indeed, domain *analysis* or *engineering* is typically a *maturation* process, where an organization that has been developing applications within a *stable* domain, decides to leverage the *commonalities* between the applications into a methodology and set of software artifacts that facilitate the development and maintenance of future applications. CXM does not fit the bill, as explained below.

First, it is not a *mature domain*, as the fragmented literature shows (see Sect. 2), with *abstractions* coming from different fields (marketing, social psychology, service design). Further, there are no easy mappings from those abstractions to IT. For example, in typical business domains, there is a relatively simple mapping from *domain concepts* to *software concepts*: real-world entities (customer, product, etc.) that the system needs to track are mapped to similarly named entities (objects, tables, etc.). Contrast that with the tortuous path of *influence factors* and how we included them in the consumer profile (see Sects. 2.1, 2.5.1, and 4.2).

A *corollary* of the lack of maturity of the CXM domain, is the absence of existing CXM applications that we could draw upon, for analysis, design, or implementations. First, while the *concept* of CXM is not new, there are no methodical industrial implementations. Most e-commerce tool vendors, like our industrial partner, recognize the potential of CXM functionalities. However, they do not offer *CXM frameworks* beyond, perhaps, standalone machine learning or text mining libraries. Industrial implementations such as Amazon's Go store are more like one-of-a-kind *technology showcases*.

7.2 Do not call us, we will call you

Typical application frameworks rely on what is called the *Hollywood Principle* to: (1) maximize the scope of the shared logic, and (2) minimize the dependencies between the user code and the framework code. Typically, a framework would define a number of *hotspots* (variation points) that correspond to *contracts* (typically, *interfaces*) that *user code has to implement* to benefit from the services offered by the framework. This way, user code does not need to invoke those services explicitly ("do not call us"); however, by "registering with the framework" (instantiating its 'hotspots'), the methods it implemented to "fulfill its contractual obligations" will get called ("we will call you"), thereby benefiting from the services offered by the framework.

This metaphor, which works well with *infrastructure services*, becomes a bit more tenuous with *domain services*, where *contracts* need to be specified in domain-specific terms. This is even more difficult with CXM functionalities, where the problem goes beyond the issue of *domain vocabulary*. We showed in Sect. 6 how we alleviated this problem by encoding some of the CXM functionalities using a *templating language* where *template variables* are defined *intensionally*. However, this does not address the bigger problem with CXM functionalities that involve machine learning, discussed next.

7.3 The machine learning conundrum

Table 1 shows a number of interactions (CMMN *tasks*) that can be initiated by the retailer at different stages of the purchasing process. Task $T13$ (*cross-selling*), initiated at the check-out or feedback stage, recommends products that are typically purchased *along with* the one being purchased or evaluated. Task $T4$ talks about *personalized recommendations* that refer to the consumer profile. It would be convenient if a CXM framework could include library functions such as:

```
Sorted<Product> findProductsPurchasedWith(Product p);
Sorted<Product> findPreferredProducts(Customer c,
                        ProductCategory pc);
Sorted<Product> findPreferredProducts(
                  CustomerCategory cc, ProductCategory pc);
```

We saw in Sect. 6 difficulties inherent in coding such functions in a domain-independent way, as discussed above (Sect. 7.2). The solution that we proposed encodes these functions in an *intentional* way, where we refer to data elements/attributes by their *semantic role* in the description of an entity, as opposed to *by name*. The example shown in Sect. 6 did a simple attribute value comparison. However, the above three functions require using *machine learning algorithms* on a *vectorial encoding* of the domain data. For example, one can imagine the lfunction *findProductsPurchasedWith(Product p)* using some form of clustering or association rule mining, on a vectorial or a set representation of a historical database of purchase transactions. In the simplest case, the implementation of this function requires two major decisions: 1) choosing an *algorithm* appropriate for the data (types of attributes, size, distribution, etc.) and task at hand (association, classification, regression, etc.), and 2) choosing an *encoding* of the input data appropriate for that algorithm. Both decisions involve a combination of machine learning expertise, domain insights, and trial and error, all of which are challenging—but not impossible to encode for reuse.

We have ongoing work to develop a *machine learning framework* aimed at *domain experts* such as marketing analysts, that helps them:

1. Translate a *domain problem*, such as "find products often bought together", into a *data analysis problem* such as "clustering" or "association rule mining";
2. Adopt or adapt a machine learning *solution process/workflow*. Indeed, typical machine learning problem solving often a sequence of tasks starting with data preparation, followed by the application of one or several machine learning algorithms, followed by data presentation. Data preparation may involve data collection, data cleaning, filtering, parameter estimation, dimensionality reduction, etc.;
3. Chose the appropriate algorithm for each step of the solution process/workflow;
4. Execute the solution on the actual data at hand;
5. Presenting or interpreting the output (data, models, etc.).

For the purposes of our CXM framework, we can focus on the *common problems* that arise in constructing data elements (see Questions Q3.2 and Q3.3 in Sect. 2.1), and on the interactions shown in Table 1, with the goal of arriving at *configurable versions* of the three functions shown above[4].

7.4 One purchase scenario versus many

The tool presented in Sect. 4 enables a marketing analyst to specify the user requirements for a *purchase scenario*; such requirements serve as the input to an MDE tool chain that ultimately produces executable CXM code. Recall that a purchase scenario consists of an *experience-enhanced purchasing process*, and the data needed for the process. In turn, the *experience-enhanced purchasing process* consists of a subset of the Bagozzi's purchasing process (Fig. 1) that is relevant to the type of purchase at hand, augmented/decorated with *experience-enhancing interactions* designed following the principles explained in Sect. 2.4. This raises the question of what is the appropriate *scope* for a *purchase scenario*.

To the extent that one purchase scenario results in one software application, with its own data model and persistence layer, the idea of one purchase scenario per retailer is appealing, due to its simplicity. This approach would probably make sense for a grocery chain, a pet food and supplies chain, or a sports equipment chain (actual customers of our industrial partner), i.e. retailers that focus on a single kind of product. Using a single scenario does not mean that we should trigger all of the interactions for every product in the store. For example, a sports equipment chain would carry products from a three-dollar tennis ball to a three-thousand dollar high-performance or electrical bicycle. The interactions shown in Table 1 can be conditioned on the price of the product of interest. We could also add transitions (see Figure 8) that skip stages based on the price of the article.

However, if we have a retailer that supports a wide variety of product lines, from six-thousand dollar home entertainment systems to tomatoes or socks (e.g. Walmart), then several purchase scenarios need to be specified. This raises two problems. The first is related to the multitude of data models, each with its definition of customer and product

[4] Interestingly, Shopify publishes one such configurable app for the "find products often bought together": https://apps.shopify.com/frequently-bought-together.

models and categories. The second is related to the multiple processes that could or should be enacted.

The first problem is relatively easy to solve: we can treat the data model specific to a scenario as a *view* on a central model, and generate a single persistence layer for all the scenarios. This entails merging the data models specific to each scenario, prior to invoking the code generator (see Fig. 4 in Sect. 3). This requires some changes to the tool to define purchase scenarios within an *enterprise container*, and support the definition of several scenarios per container; the final tool in the tool chain (code generator) would then generate the code for the scenarios within a container in a single batch.

The second problem is slightly more complex, and deals with the issue of having several process models. Notice that for the kind of immersive applications described in the introduction (Sect. 1), this would not be a big problem. Indeed, this is the very notion of *context sensitivity*, where a different application takes over as a customer moves from the appliances sector to the grocery sector. It is more complicated within the context of multi-channel or omni-channel commerce, or even online shopping where one can switch back and forth between two product lines with a single keystroke. The context switching can be dizzying. We are working with digital marketing specialists for insights as to the best strategy to handle different, possibly overlapping, experiences.

7.5 Validation

In a nutshell, our work aims at developing a *methodology* and a *software development framework* (CxDev), that *helps* retailers *develop effective* add-on customer experience management *(CXM) functionalities* to their e-commerce platforms. Our work is consistent with the design science research (DSR) methodology [16], and the *artefact(s)* that we are designing consists of a methodology and (embodied in) a software development framework meant to achieve a number of goals, set in italics in the previous statement. We can identify three distinct but related goals:

- G_1 It *helps develop* CXM functionalities. We mentioned throughout this paper that CXM is a novel *software application domain* that challenges traditional software development techniques, specifically the ones related to requirements and specifications (see Sects. 1 and 2.1)
- G_2 It pertains to *CXM functionalities*. This relates to the fact that the knowledge embodied in the methodology and the framework represents consensual, state of the art CXM knowledge,
- G_3 The CXM functionalities that are developed using the methodology and framework are *effective*.

Each one of these goals has a number of subgoals. For example, goal G_1 can be broken down into several subgoals, corresponding to help at different stages of the development lifecycle:

- $G_{1.1}$ Help business (marketing) analysts specify functional requirements for a CXM application
- $G_{1.2}$ Help developers produce executable software from such functional requirements in a target architecture and technology, with minimal effort
- $G_{1.3}$ Help produce code that exhibits different intrinsic (e.g. modularity) and run-time qualities (e.g. performance)

In turn, goal G_3 can be broken down into two distinct subgoals relating to *effectiveness* from the point of view of the consumer or the retailer:

- $G_{3.1}$ The developed functionalities enhance consumers' experiences in interacting with the retailer.
- $G_{3.2}$ The developed functionalities enable the retailer to attain his business objectives.

$G_{3.1}$ and $G_{3.2}$ are distinct but related. A user might find that an experience is 'pleasant' ($G_{3.1}$), without buying anything. When I walk into a store, I appreciate when a salesperson tells me right away that they do not have what I am looking for instead of trying to sell me something else. From the point of view of the business, effectiveness ($G_{3.2}$)may be measured in terms of: (1) maximizing the probability or the value of each transaction, or (2) developing or enhancing brand loyalty, or (3) maximizing lifetime value, i.e. the number of purchases that the consumer will make in the long run.

Given these goals, the Goal/Question/Metric (GQM) framework [12] can then be used to derive, for each goal, research questions; these questions would, in turn, lead to metrics which would guide the design of experiments to collect such metrics (see e.g. [18].

At this stage of the research, we are particularly interested in goals $G_{1.1}$, $G_{1.2}$, G_2, and $G_{3.1}$. The quality of the code generated by our MDE tool chain (see Sects. 3, 5, and 6) is not an immediate concern (goal $G_{1.3}$). We are not concerned either, about the efficacy of the CXM functionalities with regard to the business objectives of the retailers (goal $G_{3.2}$, e.g. maximizing shopping cart value versus lifetime value), for both ethical and practical[5] reasons.

With regard to goal $G_{1.1}$, we are currently working with marketing researchers and practitioners to validate the requirements elicitation tool (*tool for specifying purchase scenarios*, see Sect. 4). We are facing two methodological challenges, among others:

[5] It is impractical to design a *controlled* experiment to check whether the addition of CXM functionalities increases the frequency or average value of transaction; see e.g. [18].

1. We have the intuition that the potential users of our CXM specification tool would be marketing or *digital* marketing specialists, but we do not know what their job titles would be, in part because such 'jobs' are emerging!
2. We have no basis for comparison. We know of no CXM specification methodology or tool that we can compare ours to.

We are working with our industrial partner (an e-commerce tool vendor) to help us identify target users within its customer base (various retailers). We are also working with marketing researchers to design experiments to validate the CXM contents embodied in our methodology and tools (goal G_2).

With regard to goal $G_{1.2}$, we evaluated some parts of an earlier version of the MDE tool chain with a case study related to sports equipment [7]. We were able to validate the feasibility of the approach in general, and the code generation for the data aspects of CXM applications. The data aspects, as described in Sects. 4.2 and 5.1, have not evolved significantly since 2017. The *behavioral aspects* of the CXM functionalities are new and ongoing and remain to be validated. We should also note that, for the time being, the CXM functionalities are generated standalone. We have not yet focused on architectural integration of CXM functionalities into traditional e-commerce site functionalities. This depends, to a large extent, on the architecture of the host platform.

With regard to goal $G_{3.1}$, i.e. that the CXM functionalities developed do indeed enhance consumers' experiences, we would need an actual implementation and deployment of CXM functionalities into an operational e-commerce or omni-channel commerce platform. Our tools are not yet mature for integration into a production environment. Working with our industrial partner, we will identify a pilot site to test our CXM functionalities. Note however, that we have performed a partial validation of some of the CXM functionalities developed within the context of this project. In particular, we developed a generic recommendation algorithm that uses *veristic variables* [31,32] to take into account the variable semantics of concept properties in [9]. Indeed, a class of recommendation algorithms recommend products based on similarities between consumer properties and product properties. These properties have a multitude of semantics : monovalued vs. multivalued, certain vs. uncertain and crisp vs. fuzzy. We have shown empirically that taking into account these variabilities improves the performance of the recommendation algorithm when compared to baseline algorithms [9].

8 Conclusion

The IoT, AI, and cloud computing enable us to develop applications that provide rich and personalized customer experiences around existing e-commerce platforms; *if only* we knew *what* to develop, from a functional point of view, and incidentally, *how* to develop it by leveraging reusable components. That is the *domain engineering* mandate given to us by our industrial partner, the vendor of an e-commerce product suite.

Domain engineering is usually done as a logical step in the maturation process of an organization that accrues a deep expertise and a 'large' portfolio of applications within a relatively limited domain. By contrast, we had to contend with some *fundamental questions* about CXM, that a fragmented literature only partially addressed; we also had no existing application portfolio to start with, beyond the 'fantasies' of marketing and IT visionaries. In this paper, we discussed the main challenges, and described our strategy for addressing them. Our work is still in the early stages, and there is a lot of development work ahead of us, both in terms of methodological content and tooling. However, we hope that our general approach enables us and others to research these issues in a methodical manner.

References

1. Alegre U, Augusto JC, Clark T (2016) Engineering context-aware systems and applications: a survey. J Syst Softw 117:55
2. Ariely D, Carmon Z (2000) Gestalt characteristics of experiences: the defining features of summarized events. J Behav Decis Mak 13(2):191–201
3. Ariely D, Zauberman G (2003) Differential partitioning of extended experiences. Organ Behav Hum Decis Process 91(2):128–139
4. Azjen I (1991) The theory of planned behavior. Organisational Behav Human Dec Process 50:179
5. Bagozzi RP, Gurhan-Canliu Z, Priester JR (2007) The social psychology of consumer behavior. Open University Press
6. Bagozzi RP, Warshaw PR (1990) Trying to consume. J Consum Res 17:127–140
7. Benzarti I, Mili H (2017) A development framework for customer experience management applications: principles and case study. In: Proceedings—14th IEEE international conference on e-business engineering, ICEBE 2017
8. Benzarti I, Mili H, Leshob A (2020) Cxdev: A case study in domain engineering for customer experience management. In: Reuse in emerging software engineering practices, pp. 100–116. Springer International Publishing, Cham
9. Benzarti I, Mili H, Paillard A (2020) A content based e-commerce recommendation approach under the veristic framework. In: Chao KM, Jiang L, Hussain OK, Ma SP, Fei X (eds) Advances in E-business engineering for ubiquitous computing. Springer, Cham, pp 495–514
10. Bitran GR, Ferrer JC, Rocha e Oliveira P (2008) Om forum–managing customer experiences: perspectives on the temporal aspects of service encounters. Manuf Serv Oper Manag 10(1):61–83
11. Bock C, Zha X, Hw Suh, Lee JH (2010) Ontological product modeling for collaborative design. Elsevier, pp 510–524
12. Briand LC, Differding CM, Rombach HD (1996) Practical guidelines for measurement-based process improvement. Softw Process Improv Pract 2(4):253–280

13. Cook LS, Bowen DE, Chase RB, Dasu S, Stewart DM, Tansik DA (2002) Human issues in service design. J Oper Manag 20:159
14. Dey AK, Abowd GD, Salber D (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Comput Interact 16:97
15. Gruber T (2007) Ontologies, web 2.0 and beyond
16. Hevner A, Chatterjee S (2010) Design science research in information systems. In: Design research in information systems, pp. 9–22. Springer
17. Holmqvist J, Guest D, Grönroos C (2015) The role of psychological distance in value creation. Manag Decis 53(7):1430–1451
18. Ba Kitchenham, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, El Emam K, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734. https://doi.org/10.1109/TSE.2002.1027796
19. Kurz M, Schmidt W, Fleischmann A, Lederer M (2015) Leveraging cmmn for acm: examining the applicability of a new omg standard for adaptive case management. In: Proceedings of the 7th international conference on subject-oriented business process management, pp. 1–9
20. Lee JH, Fenves SJ, Bock C, Suh HW, Rachuri S, Fiorentini X, Sriram RD (2011) A semantic product modeling framework and its application to behavior evaluation. IEEE Trans Autom Sci Eng 9(1):110–123
21. Liu Y, Shrum LJ (2002) What is interactivity and is it always such a good thing? Implications of definition, person, and situation for the influence of interactivity on advertising effectiveness. J Advert 31(4):53–64
22. Meyer C, Schwager A et al (2007) Understanding customer experience. Harv Bus Rev 85(2):116
23. Mili H, Benzarti I, Meurs MJ, Obaid A, Gonzalez-Huerta J, Haj-Salem N, Boubaker A (2016) Context aware customer experience management: A development framework based on ontologies and computational intelligence. In: Sentiment analysis and ontology engineering, pp. 273–311. Springer
24. Motahari-Nezhad HR, Swenson KD (2013) Adaptive case management: overview and research challenges. In: 2013 IEEE 15th conference on business informatics, pp. 264–269. IEEE
25. Perera C, Zaslavsky A, Christen P, Georgakopoulos D (2014) Context aware computing for the internet of things: a survey. IEEE Commun Surv Tutor 16:414
26. Preuveneers D, Novais P (2012) A survey of software engineering best practices for the development of smart applications in ambient intelligence. J Ambient Intell Smart Environ 4(3):149–162
27. Routis I, Nikolaidou M, Alexopoulou N, Anagnostopoulos D (2018) Empowering knowledge workers with cmmn: the concept of case learning. In: 2018 IEEE 22nd international enterprise distributed object computing workshop (EDOCW), pp. 33–36. IEEE
28. Routis I, Nikolaidou M, Anagnostopoulos D (2018) Using CMMN to model social processes. In: Lecture notes in business information processing 308(February): 335–347
29. Solomon MR, Dahl DW, White K, Zaichkowsky JL, Polegato R (2014) Consumer behavior: buying, having, and being. Pearson Toronto, Canada
30. Walker B (2011) The emergence of customer experience management solutions. For eBusiness& Channel Strategy Professionals
31. Yager RR (2002) Querying databases containing multivalued attributes using veristic variables. Fuzzy Sets Syst 129(2):163–185
32. Yager RR (2007) Veristic variables and approximate reasoning for intelligent semantic web systems. In: Forging new frontiers: fuzzy pioneers I, pp. 231–249. Springer
33. Zomerdijk LG, Voss CA (2010) Service design for experience-centric services. J Serv Res 13(1):67–82