Dissertation

submitted to the

Faculty of Computer Science and Mathematics
University of Passau

A Layered Conversational Recommender System

Sven Radde

June 25, 2013

Supervisor: Prof. Dr. Burkhard Freitag

Dissertation for the acquisition of the degree of a doctor in natural sciences at the Faculty of Computer Science and Mathematics of the University of Passau.

1st reviewer:	Prof. Dr. Burkhard Freitag
2nd reviewer:	Prof. Dr. Harald Kosch

Abstract

In this thesis a new approach to building product recommender systems is introduced. By using a customer-centric dialogue, the customers' preferences are elicited. These are the basis for inferring utility estimations about the desired technical properties of the products in question. Systems built this way can both operate autonomously, e.g., in an online store, and support a salesperson directly at the point-of-sale.

The core of the approach is formed by a layered domain description that models customer stereotypes and needs, product attributes, the products themselves, and the causal interrelations between customer and product properties. Maintenance of the domain description, i.e., keeping the model up-to-date in face of frequent changes, is facilitated by the clear separation of concerns provided by the layered structure. In fact, the most frequently used class of updates can be handled in an entirely automated way if some constraints are satisfied. On a high level of abstraction, the system behavior is described by State Charts that are parameterized according to the domain description. Those parts of the system description where State Charts would be too imprecise are implemented by separate components realizing the required complex semantics. From the domain description, a Bayesian network is generated that forms the core of the inference engine of the recommender system. The network essentially controls the system-initiated dialogue flow and the recommendation process. Due to the characteristics of Bayesian networks, it is possible to respond to user-initiated dialogue steps in a natural way. Moreover, an explanation of the current recommendation can be generated without having to explicitly encode additional information in the modeling layer. Finally, a database structure and the SQL queries necessary to obtain recommendations can be inferred from the corresponding parts of the domain description. Instantiation of the system to a specific business domain is supported by a dedicated maintenance application that hides the complexities of the underlying algorithms. Thus, day-to-day system updates by non-technical domain experts, e.g., product managers, are facilitated.

The developed concepts were implemented in cooperation with a local industry partner who intends to apply the recommender system in the field of mobile communications.

Kurzfassung

In dieser Dissertation wird ein neuer Ansatz zur Konstruktion von Produkt-Beratungssystemen vorgestellt, die mit den Kunden einen bedürfnisorientierten Dialog führen und dann aus den gegebenen Antworten Abschätzungen über die jeweilige Nützlichkeit verschiedener charakteristischer Attribute der zu empfehlenden Produkte gewinnen. Die so erstellten Systeme können sowohl autonom, beispielsweise in einem Onlineshop, als auch zur Unterstützung von Verkäufern direkt im Ladengeschäft eingesetzt werden.

Eine Domänenbeschreibung, die auf mehreren Ebenen Kundenstereotypen, -bedürfnisse, Produktattribute, sowie die Beziehungen zwischen Kunden- und Produkteigenschaften und die Produkte selbst modelliert, bildet den Kern des Ansatzes. Die Pflege der Domänenbeschreibung wird durch die Unterteilung in mehrere Ebenen deutlich unterstützt. Tatsächlich kann unter gewissen Bedingungen die häufigste Klasse von Aktualisierungen vollautomatisch vom System selbst gehandhabt werden.

Mit Elementen der Domänenbeschreibung parametrisierte State Charts beschreiben das Systemverhalten auf abstrakter Ebene, während domänenspezifische Teile, welche durch State Charts nicht ausreichend präzise modelliert werden können, durch separate Komponenten implementiert werden, die die erforderlichen Funktionen realisieren. Aus der Domänenbeschreibung wird ein Bayesnetz generiert, das den Kern der Inferenzmaschine des Beratungssystems bildet und zur Steuerung des Beratungsdialogs und des Empfehlungsprozesses genutzt wird.

Die Eigenschaften von Bayesnetzen ermöglichen es unter anderem, Nutzer-initiierte Dialogschritte auf natürliche Weise in den Beratungsprozess zu integrieren. Darüber hinaus können verständliche Erklärungen für die erzeugten Produktempfehlungen generiert werden, ohne dass dazu zusätzliche Informationen explizit in die Modellierung eingebracht werden müssen. Desweiteren können die vom System zu verwendende Datenbankstruktur und die zur Empfehlungsgenerierung benötigten SQL-Abfragen aus den entsprechenden Teilen der Domänenbeschreibung erzeugt werden.

Die Instanziierung des Systems für ein konkretes Geschäftsfeld erfolgt über eine dedizierte Pflegeanwendung, die die mathematische Komplexität der eingesetzten Algo-

rithmen verbirgt und so die Aktualisierung des Systems auch durch Domänenexperten ohne technische Ausbildung, wie z.B. Produktmanager, ermöglicht.

Eine praktische Implementierung erfolgte im Rahmen eines gemeinsamen Projekts mit einem Industriepartner aus der Region, der das Beratungssystem u.a. im Bereich des Vertriebs von Mobilfunkprodukten einsetzen möchte.

Acknowledgements

My thanks, first and foremost, go to Professor Dr. Burkhard Freitag for giving me the chance of writing my dissertation at his institute, for his demanding, yet always very constructive supervision and guidance of my research, and for his patience and continued support. I would also like to thank Professor Dr. Harald Kosch for his encouraging feedback and for serving as the second reviewer of my thesis.

Furthermore, I wish to thank all my colleagues, current and past, for providing a both friendly and professional environment to work in, for the many opportunities to discuss my work with them, and for the valuable suggestions resulting from these discussions. In particular, my thanks go to Alexander Stenzer, Christian Schönberg, Christoph Ehlers, and Claudia Woller for proofreading large parts of my thesis and thus especially helping me in my fight of American vs. British English.

Also, I would like to thank the :a:k:t: Informationssysteme AG for giving me the opportunity to work with an extraordinary team in an impressive project that put many of the concepts described in this thesis into practice. Special thanks go to Bettina Zach and Swantje Leippi for our close and valuable cooperation that laid the cornerstone of this dissertation.

Finally, writing this dissertation would have been impossible without the help of my family and many friends, who supported me in a myriad of ways, big and small. I cannot name all of them personally here, but I would like to particularly express my gratitude to my loving parents and my close friend Dennis Holtschulte for their continued encouragement, most notably during a time of personal hardship.

 Part of this research was co-funded by the European Regional Development Funds (ERDF/EFRE) through the “IuK-Bayern” grant programme.

The core of our implementation of Bayesian inference is based on the SMILE reasoning engine for graphical probabilistic models, contributed to the community by the Decision Systems Laboratory of the University of Pittsburgh (`http://dsl.sis.pitt.edu/`).

Contents

1 Introduction

1.1 Aims and Objectives

The goal of this dissertation is to provide a novel method of specifying conversational recommender systems for industrial domains in which conventional recommendation techniques such as collaborative filtering cannot be used. At its most basic, the concept of a conversational recommender system is to engage in a dialogue with a prospective customer and then to select the appropriate items from a product catalogue, much as a natural salesperson would do. Figure 1.1 illustrates the basic idea of our approach at a very abstract level.

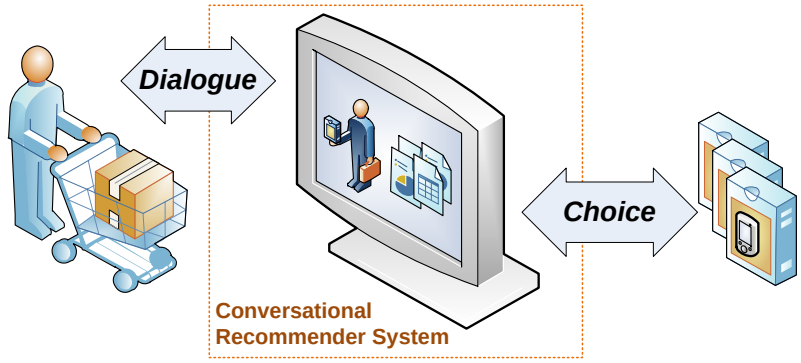

Figure 1.1: Basic idea of a conversational recommender system

A common challenge that conversational recommender systems have to meet is the necessity to maintain their internal knowledge base. The knowledge base allows them to draw conclusions from the information elicited from customers regarding the products the recommender system has to recommend. Such maintenance is particularly expensive in fast-paced industrial environments that are affected by frequent releases of new products and technological innovations.

Our approach is composed of several architectural layers. They introduce a separation of concerns that reduces maintenance complexity by keeping the adaptations necessary

to respond to changes in the domain local and allowing for automation in the most frequent cases. At the core of the layered approach lies a metamodel that allows the specification of customer and product properties and interrelations between both. From instances of this metamodel, the dialogue management and the recommendation component are generated automatically, as well as the inference engine that allows to draw the necessary conclusions from a customer's answers, as illustrated in figure 1.2.

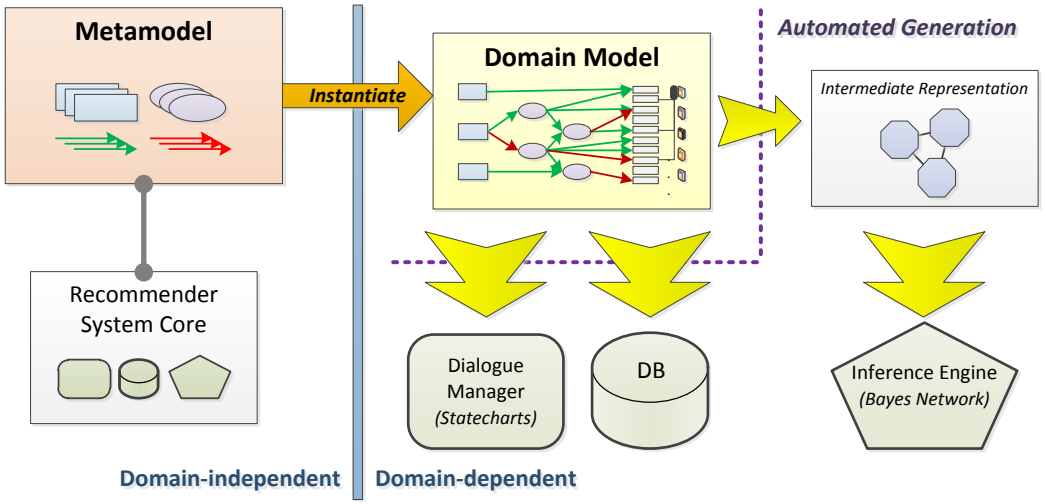

Figure 1.2: Layer structure

1.2 Motivation

Customers intending to buy a new mobile phone commonly are in significant need of qualified assistance and recommendations. Not only are the products complex by themselves, but the properties of the domain also change frequently so that possibly acquired knowledge ages quickly. Indeed, even professional salespersons find themselves challenged by the permanent need to adapt to new circumstances, which is even more pertinent if we consider not only specialised retailers but also chain stores. Competition in the market is fierce and vendors see valuable recommendations as one way to differentiate themselves from competitors.

However, due to the changes in the domain and the fact that customers buy mobile phones too infrequently to establish a realiable user profile, many traditional recommender system technologies, such as collaborative filtering, perform very poorly in

this scenario. On the other hand, more advanced recommendation approaches, such as knowledge-based and conversational systems often are not capable enough to provide a satisfying customer experience. This becomes particularly apparent, if we take into account that a suitable recommender system should also be able to support salespersons directly at the point-of-sale.

Our chosen approach combines several techniques into an integrated system that provides a solution applicable for this use case. While the individual building blocks are considered to be well-established in their respective communities, our combination of the techniques results in a novel approach to conversational recommender systems that alleviates many of the problems with knowledge-based systems for our use case. In particular, to the best of our knowledge, Bayesian networks as the central component of our inference engine, were not known to be suitable or adequate for the task at hand before our implementation.

In cooperation with a local industry partner, we completed a project to deliver an industrial strength prototype implementation of the developed concepts in a span of about two years.

1.3 Key Contributions

Using the mobile telecommunications market as an example, we have made a conversational recommender system operational in a challenging industrial domain. A metamodel-based generative approach enables efficient maintenance of the system in spite of frequent changes in the domain. The inference engine is based on a Bayesian network and functions adequately in this context. The dialogue management component is flexible enough to even allow using the recommender to assist salespersons directly at the point-of-sale, apart from the more conventional applications in webstores or kiosk computers as illustrated in figure 1.3.

In addition to implementing a research prototype based on the mentioned techniques, the developed concepts were also put into practice during a joint R&D project with our local industry partner, the :a:k:t: Informationssysteme AG in Passau. Their next generation of Point-of-Sale assistance software will be able to include recommendation technologies relying on the approach described in this thesis. As part of this software, a representative knowledge base for their use case was created and is being maintained routinely.

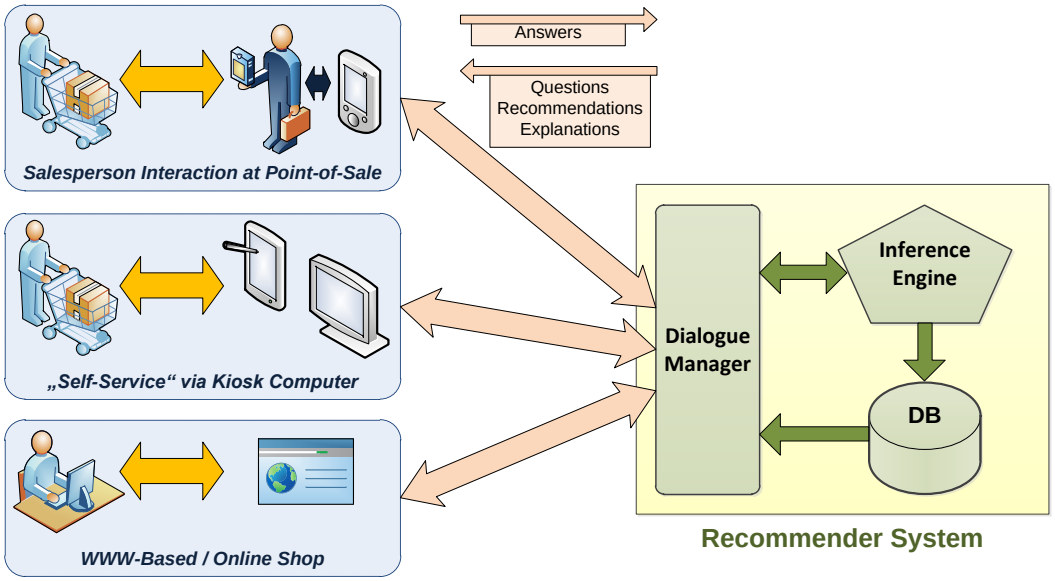

Figure 1.3: Different interaction scenarios

During this project, the general applicability of our approach as a recommender system for mobile telecommunication products was positively evaluated in a study conducted by an external market research institute, whereas further experiments with our prototype provided evidence for the transferability of the approach to other domains. Performance measurements conducted both in our lab and in a realistic deployment at our industry partner confirm that the approach adequately meets the requirements for interactive usage.

Our solution is build as a combination of a number of novel ideas:

We do not explicitly model the dialogue. Instead, we model the *domain knowledge* at a more abstract level and derive an appropriate dialogue (and other domain-dependent components) from the model. The domain model represents "soft" information that can be elicited from the customers in an *interactive dialogue* without undue cognitive effort, such as desires, needs, or expectations.

Further, the domain model represents relationships that allow inferring the "hard" information necessary to *provide recommendations* using an inference engine based on a Bayesian network. As part of the inference engine, *explanations* can be provided regarding the system behaviour and the generated recommendations.

Based on these ideas, we characterize our key contributions as follows:

- Our work describes a *novel architecture for building conversational recommender systems* based on a domain (meta-)modelling that allows using electronic recommender systems in application domains that are currently unreachable for more conventional approaches.
- Our system relies on a *new approach to elicit customer preferences* based on a flexible, model-derived recommendation dialogue, which serves to reduce the maintenance efforts commonly associated with knowledge-based recommender systems.
- The inference engine used by our approach *combines uncertain reasoning with database query technology.*
  - We show that Bayesian networks, appropriately generated from our domain model, are an adequate end efficient formalism to this end.
  - In particular, our approach uses a variant of the "Noisy-OR" method to generate the probability tables of the Bayesian networks. We chose and adapted this variant to work particularly well in our chosen use case.

1.4 Overview

The remainder of this thesis is organized as follows: Chapter 2 will provide background knowledge about the core concepts used in the thesis and give a general overview about relevant literature in doing so.

We then define our metamodel in chapter 3 and show its implementation as a Bayesian network in chapter 4. Our statechart-based approach to dialogue management is presented in chapter 5, while chapter 6 shows the usage of the Bayesian network as central inference engine of our recommender system. In chapter 7 we describe the recommendation algorithm and its implementation in SQL, before detailing system maintenance processes in chapter 8.

We provide an evaluation of the adequacy and performance of our approach in chapter 9, before concluding with pointers to future work in chapter 10 and a summary in chapter 11.

2 Background and Literature

2.1 Recommender Systems

In the broadest sense, recommender systems are software systems that assist their users in making decisions. Research in this field can be classified as a (practically-oriented) sub-discipline of artificial intelligence but can also be seen as a multi-disciplinary field due to its numerous, often commercial applications.

This section is based on the introductory chapters of [JZFF11] by Jannach et al. For deeper studies of the entire field of recommender systems, the interested reader is referred to their book along with [RRSK11].

The most prominent use case for recommender systems, and also the focus of this dissertation, is to recommend suitable retail products to potential buyers. However, techniques from the field have also been deployed to locate agreeable services (e.g., restaurants [Bur07], travel offers [RD06, Ric02], TV programmes [Höl11], or education courses [SB11]), find 'interesting' documents [GBH09, HHHC04] (i.e., to solve essentially an information retrieval task) and also to help decide about courses of action in areas like, e.g., preservation of digital heritage objects [Kul09].

Though diverse, all these applications share the common goal of recommender systems: *Present to the user the "best" alternative from a set of choices.* To put our approach into the context of the field, we modularize recommender systems into the following building blocks:

- *A method to elicit profiles*:
  Recommender systems must have means to obtain profiles from their users. These can be categorized into *implicit* approaches (e.g., observing the users' behavior when surfing a website or analyzing buying histories) or *explicit* models (e.g., requesting item ratings from users or asking questions). Some systems also combine techniques from both categories (e.g., enhance buying histories with explicit ratings) to maximize the information available for the recommendations.

- *A component to build preferences*:
  Given the profile, the recommender system must attempt to deduce the preferences of the customer. This can be done in a variety of ways and is often tied closely to the kind of the recommendation algorithm the system uses. Prevalent ways to accomplish this are, for example, comparisons to similar customer profiles or the deduction of personality traits that have not been asked for explicitly.

- *A notion of optimality*:
  Once suitable preferences have been built, the available items must be sorted accordingly. This commonly requires the use of utility functions (i.e., assigning a score to every item that reflects its usefulness), similarity measures (e.g., to find products similar to a "perfect" item), or by defining a set of constraints that optimal items must satisfy.

Regarding this classification, we can characterise our chosen approach as follows: The *profiles* are elicited *explicitly*, by asking questions in the course of an interactive dialogue. For our *preferences*, we derive importance and usefulness estimations about the technical properties of the products based on an expert-built knowledge base. These preferences are then used in a utility function to find the *optimal* products in the catalogue.

2.1.1 Collaborative Recommender Systems

A multitude of recommender systems is built around the concept of analysing the similarity of the customer's profile to other profiles. The profiles are commonly built implicitly, e.g., from buying histories or website logs. The central idea is that a customer will prefer items that similar users have preferred in the past (this "collaborating" of different users gave the approach its name). Explicit techniques to build a profile exist, e.g., by showing items to a customer and asking for ratings.

Recommender systems based on this technique are widely deployed in industry. For example, the online retail store Amazon uses collaborative filtering for its recommendations. The topic is also well-investigated in the research community (cf. [SK09, HKTR04] for just two overview-orientated works), with a particular interest being sparked a few years ago by the "Netflix Prize" competition[1]. In October 2006, the online video rental store Netflix published a transaction dataset from its log files and promised to award one million dollars to the team of researchers that would be able to improve the company's own prediction algorithms by 10%. The facts that the de-

[1] `http://www.netflixprize.com/`

manded increase of 10% can be called somewhat modest and that it nevertheless took almost three years until the prize could be claimed [Kor09], testify about the maturity of the field.

Collaborative recommender systems have another nice property, namely that they can be totally ignorant about the products they recommend. They only consider which products were sold to which customers and they do have no need at all to analyse the products themselves. For example, this property allows cross-domain recommendations if one can assume a general transferability of the customer profile across these domains (e.g., Amazon would be able to recommend CDs based on the books you bought).

However, this strength is also indicative for the greatest weakness of collaborative recommender systems: Newly appeared products that have not been sold yet, consequently are not recommended at all by such a system. This situation can be alleviated by considering similarity between products in addition to similarity between users. Called "item-based" recommendations (or item-based filtering), this approach works essentially by letting products "inherit" the rankings from other similar products, making them instantly available for recommendations. Item-based filtering only works as long as suitable similarities between products can be established.

While it is possible to build purely item-based recommender systems, the technique is commonly combined with collaborative filtering. Such systems are then called *"hybrid" recommenders*. In fact, we would assume that the majority of deployed recommender systems can be classified as "hybrid" instead of using purely collaborative or item-based techniques.

2.1.2 Content-Based Recommender Systems

Content-based recommender systems, as the name suggests, take a different approach to recommendation than collaborative filtering approaches. They basically rely on explicit information about products, such as descriptions and technical properties, and attempt to derive importances for these properties from user profiles, which are then used to recommend products. For example, when buying books, the customer profile could hold information about which genres the customer prefers and the recommender system could then recommend books of the same genre. Profiles can also be built explicitly by asking a customer the appropriate questions.

The approach does not need a large user base to work well, since a profile is individual for each customer and does not have to be compared to other profiles. Also, new

products can be recommended immediately once their properties are known. On the other hand, content-based recommenders have to face challenges dealing with new customers for whom no profile was built yet, because the design of the system generally centers around the question of how products can be classified as compatible with a user's profile (the direct correlation of the genre in the example above is strongly simplified).

Apart from the retail scenario, the approach is often used in information retrieval or information filtering, respectively, to select documents relevant for a user. The nice property of this use case is the relative simplicity of automatically extracting all meaningful item properties directly from the documents.

2.1.3 Knowledge-Based Recommender Systems

Both collaborative and content-based recommender systems rely on a user profile which is commonly gathered implicitly, i.e., by observing the customer's actions such as his/her purchases over a longer period of time. However, in many domains such profiles are not available since they involve a lot of "one-time" buyers with very few interactions with the store and the recommender system. In those cases, a collaborative recommender system will not perform well [Bur00]. For example, german customers buy a new mobile phone roughly every two years (the common term of a subscription contract), so that their profile, even if any could be built, would likely be out-of-age by the time of their next interaction with the recommender system. Also, consider that it may not be possible to re-identify customers in such scenarios, much less if we take "offline" use cases into account.

Thus, we need recommender systems that can elicit a profile quickly, which commonly means to ask explicit questions about information that the customer can provide. This often excludes technically oriented questions as would be necessary for content-based recommenders, as customers cannot be expected to be able to answer them in the case of more complex products. Because of the interactive style of eliciting the profile and providing recommendations, such systems are also called *conversational* recommender systems.

The conclusions about the properties of recommendable products must then be drawn from combining the profile with an additional knowledge base, also called a domain model. A domain model can for example contain "recommendation rules" that match customer properties to product properties or more direct similarity rules between user profiles and products.

Knowledge-based recommender system can commonly be classified as constraint-based systems that attempt to solve a constraint-system which is created from the user profile based on the possible solutions given by the product catalogue, or as case-based systems that compare the items in the product catalogue with a hypothetical "optimal" product in terms of similarity. In both cases, the customer may be required to change his/her preferences if the recommendation algorithm did not find appropriate products.

As mentioned, knowledge-based recommenders are able to avoid the "cold-start" problems prevalent in other approaches. However, they do so at the cost of having to maintain the knowledge base, including the recommendation rules and the necessarily extensive information about the products themselves.

2.2 Probabilistic Reasoning

This section is based primarily on [RN10] and at times borrows from a lecture script at our chair [Fre10, ch. 8]. The reader is referred to these works for a more exhaustive treatment of probabilistic and particularly Bayesian reasoning. [Düm03] may be taken as a textbook-style introduction into the topic, too.

2.2.1 General

Probabilistic reasoning must be employed if the domain to investigate contains uncertainty, which would make classical logical reasoning infeasible. A logical agent in worlds with uncertainty would have to consider every logically possible *belief state*, making rational choices impossible, or, at least, practically useless. There are three basic reasons for the presence of uncertainty in domain modelling (quoted from [RN10, 13.1.1]):

- *Laziness:*
  It is too much work to list the complete set of antecedents and consequents needed to ensure exceptionless rules and it would be too hard to use such rules.
- *Theoretical ignorance:*
  Science has no complete theory for the domain.
- *Practical ignorance:*
  Even if we know all the rules, we might be uncertain about a particular situation

because not all necessary tests have been run or could be run.

In order to support uncertainty, agents must extend their belief state model by the notion of a *degree of belief*, enabling them to use probability theory and utility theory to extend their reasoning ability to *expected* outcomes in a system of *random variables* with their associated probability distributions (also called a *probability model*).

We will now introduce some basic definitions and notations (based on [RN10, 13.2]).

Definition 2.2.1 (Random Variable) A *random variable* has a *name* that begins with an uppercase character, e.g., *Var* and an associated *domain*, which is a *set of values* that the variable can take on, denoted as, e.g. $dom(Var) = \{val_1, val_2, val_3\}$.

Elementary propositions involving random variables would then be written as, e.g., $Var = val_1$ and the common connectives of propositional logic can be used to create more complex propositions, e.g., $Var = val_1 \vee Var = val_2$.

If the domain of a random variable is Boolean, i.e., $dom(Var) = \{true, false\}$, we may use the shorthand notations of *var* and $\neg var$ for $Var = true$ and $Var = false$, respectively. Also, we may denote, e.g., val_2 as an abbreviation for $Var = val_2$, if doing so does not create ambiguity.

An important note about infinite domains While random variables may have infinite domains in general, we limit ourselves to *finite, explicitly enumerated domains* for the purposes of this thesis. In many of the cases that we cover here, even Boolean variables will suffice.

Definition 2.2.2 (Unconditional Probability) Propositions that involve random variables have an assigned *unconditional probability* $P(\pi)$ for a proposition π, representing the probability that π evaluates to *true* given all possible worlds.

It holds that $0 \leq P(\pi) \leq 1$.

Also, given a random variable Var, it holds that $\sum_{v \in dom(Var)} P(Var = v) = 1$.

Definition 2.2.3 (Probability Distribution) As an abbreviation for the unconditional probabilities of *all* the possible values of a random variable Var with a finite domain $dom(Var) = \{val_1, \ldots, val_n\}$ having a pre-defined ordering, we write its *probability distribution* as $\mathbf{P}(Var)$. The result of $\mathbf{P}(Var)$ is a vector as follows:

$\mathbf{P}(Var) = \langle P(val_1), \ldots, P(val_n) \rangle$

It is also possible to denote a combined probability distribution for several random variables, e.g., $\mathbf{P}(A, B, C)$, which lists the probabilities for every possible combination of the values of A, B, and C. Such a distribution is called the *full joint distribution* and often written in tabular form, in this case called the full joint distribution table.

2.2.2 Bayesian Inference

The full joint distribution table introduced in definition 2.2.3 is the most powerful instrument for inference in a system of random variables. It enables the computation of all desirable marginal probabilities and probability distributions by simply summing up the appropriate table entries. Unfortunately, as the size of the table grows exponentially with the number of random variables, using the full joint distribution is infeasible for many practical problems (e.g., anticipating our use case, we will have to deal with more than 100 random variables).

If a given system of random variables conforms to certain constraints, Bayesian networks form a more efficient representation of the full joint distribution in practice. In order to introduce Bayesian networks, we need to establish some more definitions, leading to *Bayes' Law* and the concept of *conditional independence*:

Definition 2.2.4 (Conditional Probability) Let π and κ denote two propositions. We define the *conditional probability* of "π given κ" $P(\pi|\kappa)$ as follows:

$P(\pi|\kappa) = \frac{P(\pi \wedge \kappa)}{P(\kappa)}$, whenever $P(\kappa) > 0$.

Similarly to definition 2.2.3, when we consider two random variables A and B with finite domains, a *conditional probability distribution* $\mathbf{P}(A|B)$ can be used as an abbreviation over listing the conditional probabilities $P(A = a_i|B = b_j)$ for each possible i, j pair.

Definition 2.2.5 (Product Rule) We can write the definition of conditional probability in a different form, called the *product rule*:

$P(\pi \wedge \kappa) = P(\pi|\kappa)P(\kappa)$, which can be equivalently formulated as

$P(\pi \wedge \kappa) = P(\kappa|\pi)P(\pi)$

Definition 2.2.6 (Bayes' Law) By equating both forms of the product rule, we define *Bayes' Law*:

$P(\kappa|\pi) = \frac{P(\pi|\kappa)P(\kappa)}{P(\pi)}$

For two random variables A and B, Bayes' Law can be rewritten using probability distributions:

$\mathbf{P}(B|A) = \frac{\mathbf{P}(A|B)\mathbf{P}(B)}{\mathbf{P}(A)}$

If the **P**-notation is used in this way, the expression is to be interpreted as a set of equations with one element for each possible combination of the values of the involved random variables (i.e., the above expression is a shorthand notation for writing a set of $|dom(A)| \cdot |dom(B)|$ individual equations $P(B = b_j|A = a_i) = \frac{P(A=a_i|B=b_j)P(B=b_j)}{P(A=a_i)}$).

The meaning of Bayes' Law becomes more apparent, if we replace A and B by other, more intuitive variable names, *Cause* and *Effect*:

$$\mathbf{P}(Cause|Effect) = \frac{\mathbf{P}(Effect|Cause)\mathbf{P}(Cause)}{\mathbf{P}(Effect)} \tag{2.1}$$

This may still not seem particularly helpful at first, but it is often the case that good probability estimations for three of the probability distributions exist. If, e.g., we have knowledge about the absolute probablities of causes and effects and we know how causes induce effects, we can make conclusions about the likelihood of a cause if we observe some effects in a particular case (this is often called the *"diagnostic" view*).

Also, Bayes' Law enables us to combine several effects. Consider the following expression that contains a combination of two different effect-variables:

$$\mathbf{P}(Cause|Effect_a, Effect_b) = \frac{\mathbf{P}(Effect_a, Effect_b|Cause)\mathbf{P}(Cause)}{\mathbf{P}(Effect_a, Effect_b)} \tag{2.2}$$

If this formula is used to calculate the complete conditional probability distribution $\mathbf{P}(Cause|Effect_a, Effect_b)$, we would obviously have to know the entire conditional probability distribution for $\mathbf{P}(Effect_a, Effect_b|Cause)$. While this may be feasible for a small number of effects, it does not scale much better than directly using the full joint distribution.

For a more efficient way to calculate $\mathbf{P}(Cause|Effect_a, Effect_b)$, we therefore need to make some more assertions that allow us to simplify the expression further.

Definition 2.2.7 (Conditional Indepence) Let A, B, and C denote random variables. A and B are *conditionally indepent* given C, iff

$\mathbf{P}(A, B|C) = \mathbf{P}(A|C)\mathbf{P}(B|C)$, which is equivalent to

$\mathbf{P}(A|C) = \frac{\mathbf{P}(A,B|C)}{\mathbf{P}(B|C)}$ and $\mathbf{P}(B|C) = \frac{\mathbf{P}(A,B|C)}{\mathbf{P}(A|C)}$.

In other words, the conditional probability of "A given C", may not depend on the state of B (and vice versa).

If we assume that Effect_a and Effect_b are conditionally independent given Cause, we can use this independence to rewrite equation 2.2 as:

$$\mathbf{P}(\mathit{Cause}|\mathit{Effect}_a, \mathit{Effect}_b) = \frac{\mathbf{P}(\mathit{Effect}_a|\mathit{Cause})\mathbf{P}(\mathit{Effect}_a|\mathit{Cause})\mathbf{P}(\mathit{Cause})}{\mathbf{P}(\mathit{Effect}_a, \mathit{Effect}_b)} \tag{2.3}$$

It is common to regard the entries of $\mathbf{P}(\mathit{Effect}_a, \mathit{Effect}_b)$ as "normalisation factors" which ensure that the corresponding entries in $\mathbf{P}(\mathit{Cause}|\mathit{Effect}_a, \mathit{Effect}_b)$ (i.e., single "rows" of the vector/table) sum up to 1. So we can replace $1/\mathbf{P}(\mathit{Effect}_a, \mathit{Effect}_b)$ by α in the notation (often, the α is also left out altogether, as the normalisation can be accomplished without knowing the factor, based alone on the probability distribution to be normalised):

$$\mathbf{P}(\mathit{Cause}|\mathit{Effect}_a, \mathit{Effect}_b) = \alpha\mathbf{P}(\mathit{Effect}_a|\mathit{Cause})\mathbf{P}(\mathit{Effect}_b|\mathit{Cause})\mathbf{P}(\mathit{Cause}) \tag{2.4}$$

Product rule and conditional independence can also be combined to give us a representation of the full joint distribution:

$$\begin{aligned} \mathbf{P}(\mathit{Effect}_a, \mathit{Effect}_b, \mathit{Cause}) &= \mathbf{P}(\mathit{Effect}_a, \mathit{Effect}_b|\mathit{Cause})\mathbf{P}(\mathit{Cause}) && (2.5)\\ &= \mathbf{P}(\mathit{Effect}_a|\mathit{Cause})\mathbf{P}(\mathit{Effect}_b|\mathit{Cause})\mathbf{P}(\mathit{Cause}) && (2.6) \end{aligned}$$

The equation can be generalised to n conditionally independent effects, as follows:

$$\mathbf{P}(\mathit{Cause}, \mathit{Effect}_1, \ldots, \mathit{Effect}_n) = \mathbf{P}(\mathit{Cause}) \prod_{i=1}^{n} \mathbf{P}(\mathit{Effect}_i|\mathit{Cause}) \tag{2.7}$$

A full joint probability distribution specified in this way is commonly called a *naïve*

Bayes model, because it is frequently used without proving the necessary conditional independence of the effect variables. In fact, such a model is often used when it is even known that the variables are *not* actually independent [RN10, p. 499] and it is found that the approach works well nonetheless [MN98, Fri97, DP96].

The principal advantage of specifying a full joint distribution in this way is the fact that the tables needed to represent the conditional probability distributions are much smaller than the one table holding the full joint distribution would be. Entries from the full joint distribution can then be obtained by making on-demand calculations based on equation 2.7.

2.2.3 Bayesian Networks

A Bayesian network is a data structure that extends equation 2.7 to represent any full joint probability distribution, exploiting possible conditional independencies between random variables to reduce the size of the necessary probability tables.

Definition 2.2.8 (Bayesian Network) A *Bayesian Network* is a directed acyclical graph (DAG) with the following properties:

- Each node corresponds to a random variable with a finite domain. If there is an edge from node X to node Y, X is a *parent* of Y and the set of parent nodes of Y is denoted as $Parents(Y)$. The edge is said to represent a "direct influence" from X to Y.
- Each node must be conditionally independent of its non-descendants, given its parents.
- Each node X_i has a conditional probability distribution $\mathbf{P}(X_i|Parents(X_i))$ that quantifies the effects of the parents on the node.

(For brevity of notation, we use the term "node" also in the meaning of "random variable corresponding to the node", if the context does not require a stricter distinction.)

We can use the Bayesian network to give us the full joint probability distribution for its variables $X_1, \ldots, X_n$ as follows:

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i|Parents(X_i)) \tag{2.8}$$

If we number the nodes $X_1, \ldots, X_n$ in such a way that $Parents(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$, we can use the so-called *chain rule* that builds on the product rule (definition 2.2.5) to give a slightly different form of equation 2.8:

$$\begin{aligned} \mathbf{P}(X_1, \ldots, X_n) &= \mathbf{P}(X_n | X_{n-1}, \ldots, X_1)\mathbf{P}(X_1, \ldots, X_{n-1}) && (2.9) \\ &= \mathbf{P}(X_n | X_{n-1}, \ldots, X_1)\mathbf{P}(X_{n-1} | X_{n-2}, \ldots, X_1) \cdots \mathbf{P}(X_1) && (2.10) \\ &= \prod_{i=1}^{n} \mathbf{P}(X_i | X_{i-1}, \ldots, X_1) && (2.11) \end{aligned}$$

From equations 2.8 and 2.11 it follows for every variable X_i in the network that:

$$\mathbf{P}(X_i | Parents(X_i)) = \mathbf{P}(X_i | X_{i-1}, \ldots, X_1) \tag{2.12}$$

In other words, the chain rule proves that the full joint probability distribution can be reconstructed from the conditional probability distributions for all X_i given all respective *predecessors* in the node ordering. Equation 2.12 now allows us to minimise the set of *parents* (i.e., the set of nodes with edges towards X_i) to those predecessors that have an actual influence on X_i. In turn, this leads to a minimal size of the conditional probability tables.

Example 2.2.9 (The "Alarm" Bayes Network) Let us complete this section by a well-known example that illustrates probability calculations relating to a freshly installed burglary alarm. (The example originates from [Pea97] whose author lives near Los Angeles, which explains the explicit interest in earthquakes.)

Your new alarm triggers (variable A) on burglaries (variable B) and also on minor earthquakes (variable E) that are common in your region. You have two neighbours, John and Mary, that will phone you if they hear the alarm (you are rarely at home since you work hard to finish your dissertation). John calls you (variable J) quite reliably when the alarm sounds but also, at times, only imagines that he has heard the alarm. Mary (variable M), on the other hand, often listens to loud music and might not hear the alarm at all.

There are a couple of additional assumptions that allow us to efficiently model this scenario in a Bayesian network. Firstly, we assume that burglaries are not related to earthquakes in any way and that John and Mary do not communicate with each other about whether they hear the alarm. Secondly, we assume that John and Mary

do neither directly observe the burglary (only via the alarm) nor do they feel the earthquakes. They are too minor to be felt by humans but tend to trigger sensitive equipment like alarms.

We can use these conditional (in-)dependencies to create a Bayesian network that captures the described scenario as shown in figure 2.1.

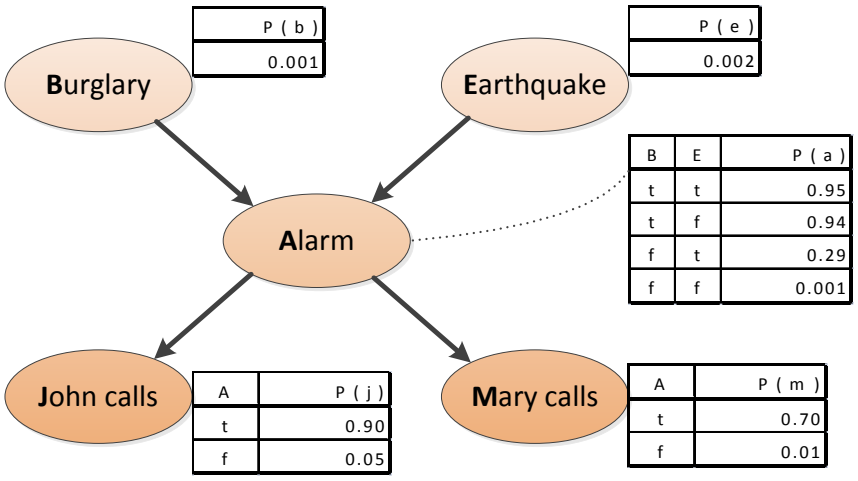

Figure 2.1: The "Alarm" network, including conditional probability tables

This Bayesian network can be used to calculate any particular entry of the full joint distribution by multiplying the appropriate conditional probabilities. For example, what is the probability that the alarm rings, but neither a burglary nor an earthquake have occured and both John and Mary call?

Using a simplified notation for the variables, we get:

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times (1 - 0.001) \times (1 - 0.002) \\ &= 0.000628 \approx 0.06\% \end{aligned}$$

By summing up the appropriate entries, any query that can be answered by the full joint distribution can be answered by using the Bayes network. As an example, the reader is encouraged to calculate the marginal probability that a burglary actually occured if both John and Mary call, i.e., $P(j \wedge m \wedge b)$. The next section introduces a tool that will make this very comfortable.

2.2.4 Bayesian Network Inference Algorithms

In this section, we give an overview about algorithms for calculating the posterior probability distributions in a Bayes network, focussing on those that are supported by the SMILE software library. SMILE ("Structural Modeling, Inference, and Learning Engine") is a C++ library developed by the Decision Systems Laboratory at the University of Pittburgh, implementing graphical probabilistic and decision-theoretic models, such as Bayesian networks, influence diagrams, and structural equation models [LV07].

The library can be used as an API from own software (bindings for Java and .NET are provided) or via the application GeNIe ("Graphical Network Interface") that is stated to have received a wide acceptance within both academia and industry[LV07]. Both GeNIe and SMILE are available free of charge and may be used for commercial purposes[2], although they are not open source.

We limit ourselves to presenting the most significant properties of the algorithms for our use case and refer the reader to the primary literature for in-depth descriptions of the implementations. The descriptions contain work that was aggregated in a master thesis at our chair [Alt09].

2.2.4.1 Exact Inference

Exact inference in Bayesian networks is principally done using the so-called "Clustering" algorithm. The algorithm is based on the idea of transforming a Bayesian network into a polytree by combining some nodes into "cluster nodes". Inference for Bayesian networks in polytree form can be done in linear complexity. However, transforming an arbitrary Bayes network into a polytree is, in the worst case, of exponential time and space complexity [RN10], leading to an exponential overall complexity, as would be expected.

The Clustering algorithm profits from networks with much evidence, since this allows a reduction of the size of the conditional probablity tables by filling in evidence variables as fixed values, which is particularly helpful for the commonly large cluster nodes.

Unfortunately, the Bayesian networks used in our approach are not very suitable for transformation into a polytree, requiring very large cluster nodes. Given real problem sizes, the algorithm would use more memory than available. Using the clustering algo-

[2] `http://genie.sis.pitt.edu/license.html`

rithm only becomes feasible after a significant amount of evidence has been gathered [Alt09].

2.2.4.2 Approximative Algorithms

Approximative algorithms are used whenever exact inference becomes infeasible because of its complexity – as it is for example the case in our approach. The algorithms are all based on the idea of estimating the posterior probability distributions by collecting a large number of "samples" from the Bayesian network, i.e., they conduct a Monte Carlo simulation. As such, their complexity is linear in relation to the number of nodes / random variables in the network. However, it requires a few thousand sampling runs to obtain a confident estimation.

Forward Sampling The most simple sampling algorithm is *Forward Sampling* as introduced in [Hen88], also called *Probabilistic Logic Sampling*. It operates on a topological ordering of the random variables. Starting with the variables that have absolute probability distributions from the beginning, one of the outcomes is chosen randomly as a "sample" based on its probability distribution. The sample is then inserted into the dependent random variables, in turn enabling random choices for their outcomes.

The posterior probability distributions per random variable are then estimated based on number of samples for each outcome. Most notable about Forward Sampling (as implemented in SMILE) is the fact that existing evidence is ignored for the sampling process. Only after a sampling run, a check whether the sample conforms to the evidence is executed, leading to a possible discard of the sample. In effect this may lead to very many discards if the Bayesian network contains much evidence, leading to very few acceptable samples which then dominate the calculation – or, indeed, to no suitable samples, which constitutes an inference failure as we frequently observed in our use case. On the plus side, the algorithm is very fast, due to its simplicity.

Most other sampling algorithms are variations on the general approach presented by Forward Sampling, introducing heuristics to weigh individual samples, to improve the sampling order, or similar.

Likelihood Weighting The *Likelihood Weighting* algorithm [FC90, RN10] fixes the values for evidence variables, thus ensuring that all samples are consistent with the evidence, contrary to plain Forward Sampling. In addition, the samples are weighted according to their likelihood in respect to the evidence. While a good idea at first,

the estimation quality suffers with many evidence variables as very many samples are assigned an infinitesimally small weight and the estimations are then dominated by a few sample runs with larger weights.

Importance Sampling This describes a family of sampling algorithms that use a modified Likelihood Weighting to better approximate the posterior distributions. They all use the concept of a so-called "importance distribution" P' and primarily differ in its exact definition.

Self-Importance Sampling [SP90] uses a weight function to constantly update its importance distribution. From the same authors, *Heuristic-Importance Sampling* uses a modified polytree algorithm to calculate likelihood functions for all unobserved (i.e., non-evidence) random variables, having to do an approximation of the likelihood function if the network does not form a polytree.

Adaptive Importance Sampling (AIS) [CD00] and *Evidence Pre-propagation Importance Sampling (EPIS)* [YD03] use even more refined methods to model their importance distributions. They are considered to be the most advanced general purpose sampling methods available, but, because of the significant calculation overhead for each sample, they are also amongst the slowest.

Backward Sampling As its name suggests, *Backward Sampling* [FdF94] uses a different sorting of the random variables than the previously described "forward" methods: Beginning from the evidence variables, the samples are collected *against* the direction of the edges towards the parent variables. Since our approach tends to introduce evidence into the Bayesian network starting at the "root" nodes and then continues to elicitate evidence following the direction of the edges, this algorithm will in fact behave similar to the normal Forward Sampling approach for our use case.

Markov-Chain Sampling Also called *Gibbs Sampling* in the case of Bayesian networks, *Markov-Chain Sampling* [RN10, 14.5.2] is another approach to approximate posterior probability distributions. Its idea to generate samples is different from the other algorithms, in that it continually makes changes to the respective preceding sample instead of generating completely new samples. This way, the algorithm settles into a "dynamic equilibrium" that represents the posterior distributions. Since the SMILE software library does not contain an implementation of the technique, we decided not to investigate it further in the course of our work.

2.3 Statecharts

We use statecharts to formalise and visualise the behaviour of our recommendation dialogue. Regarding syntax and semantics, we follow the definitions of R.J. Wieringa which suit our requirements well. The interested reader is referred to chapter 12 of [Wie03] for a more exhaustive treatment of the topic. Possibly more prevalent are UML state diagrams (cf., e.g., [BD04, 2.4.4]), that show some minor differences when compared to statecharts, primarily in syntactic details.

2.3.1 Mealy Diagrams

Since statecharts are an extension of Mealy diagrams, we will begin by illustrating the principal syntactic and semantic elements of the latter (cf. [Wie03, 12.1]).

The primary modelling elements of Mealy diagrams are *states*, represented by rounded rectangles, and *state transitions*, represented by arrows and annotated with a label of the form `event expression [guard expression] / action expression`.

A Mealy diagram must contain a starting state, represented by a bullet, and may contain final states, represented by a bullet in a circle (see figure 2.2 below). In addition to states, the diagrams may make use of arbitrary global variables (cf. [Wie03, 12.2]).

- The `event expression` is a *named event* (e.g., "answer received") that can be fired in the modelled system. All events are *broadcast* through the entire system, i.e., there is no "scoping" of events. Alternatively, *temporal events*, such as "after 5 minutes" or "at 12:00" could be specified in Mealy diagrams, but our approach does not use such events.
- The `guard expression` denotes a *condition* that may contain the common Boolean operators to combine more elementary conditions over arbitrary global variables present in the modelled system.
- The `action expression` denotes a set of *actions* to be executed. Our approach generally uses only single actions, but multiple actions could be separated by commas or semicolons to specify simultaneous or sequential execution, respectively.

We refer to these labels as "ECA" rules (for **E**vent-**C**ondition-**A**ction, as we commonly only have elementary conditions for our guard expression). Figure 2.2 shows two states S_1 and S_2, connected by the ECA rule $e[c]/a$. This modelling means that the system

will make a transition into state S_2, iff the system is currently in state S_1 and the condition c evaluates to *true*. In this case, the action(s) specified by a will be executed during the transition. Note that the formalism does not specify a general means to resolve ambiguities, i.e., if several transitions would be eligible to be executed. It is up to the modeller to avoid those ambiguities or to design an application-specific way to resolve them.

Figure 2.2: Basic Mealy diagram for $S_1 \xrightarrow{e[c]/a} S_2$

All expressions within a transition label are optional, the minimal label would be "`/`" or, depending on notation preferences, "`[]/`". No specified `event expression` means that the transition is executed immediately if the condition evaluates to *true* (one may call this a "condition change event"), whereas an omitted or empty `guard expression` can simply be regarded as shorthand for `[true]`. If no action is specified, the transition is just executed without triggering other behaviour.

A special type of state, the so-called *decision state*, represented by a hexagon, can be used to model the evaluation of a specific condition. To symbolise the fact that the system cannot remain in this state, transitions outgoing from a decision state often do not contain an `event expression` in our diagrams, as illustrated by figure 2.3. To the same end, it should be ensured that one of the `guard expression` evaluates as *true* in all cases. Our approach uses decision states at some points to make the important decisions of the system more explicit.

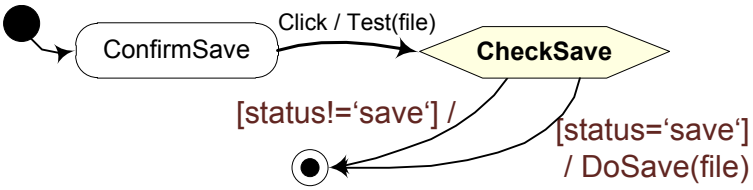

Figure 2.3: Example Mealy diagram for saving a file, showing a decision state

As a side note, we would like to observe that the Mealy diagrams shown in figures 2.2 and 2.3 of course also display valid statecharts, which merely extend the Mealy diagram syntax as decribed below.

2.3.2 Extension to Statecharts

According to [Wie03, 12.3], statecharts provide three additional modelling techniques that are not present in Mealy diagrams:

- **State reactions** allow to specify actions that are supposed to be executed when the control flow *enters* or *leaves* a particular state (in addition to actions implied by the transition ECA-rule itself). Also, other events may be handled by actions without triggering a state transition (i.e., the control flow stays within the current state). Our modelling does *not* use state reactions.
- **State hierarchies** allow to specify a system's behaviour *within a state* in a fine-grained way, essentially by allowing to recursively embed entire *sub-statecharts* into states. We use this technique to model the system's behaviour at different abstraction levels.
- **Parallelism** allows to specify that the system is in several states *simultaneously* that may then react to events *independently* of each other. In our modelling, the recommendation generation and the dialogue management are two parallel processes that we model with this technique.

State hierarchies can be used to provide statecharts for different abstraction levels of the system behaviour. As figure 2.4 illustrates, any state may itself contain another statechart, even recursively. From a more abstract point of view, the nested statechart may be seen as a "black box" and ignored. The syntax is also called "OR-state" since the control flow can only be in one of the states of the sub-chart. Labels for OR-states are frequently either omitted or moved to the top as shown in the figure.

Contrastingly, "AND-states" enable parallelism by allowing the control flow to be in several states *at once*. An AND-state contains several areas separated by dotted lines. Each of these areas contains a sub-statechart that is executed independently of the other charts. Note that there are no direct transitions between different sub-statecharts within an AND-state, but remember that events are broadcast through the entire system and therefore can be used to communicate between parallel processes. In order to specify precisely how the AND-state should be entered, *hyperedges* that target several states in different areas of the AND-state can be used, as shown in figure 2.5.

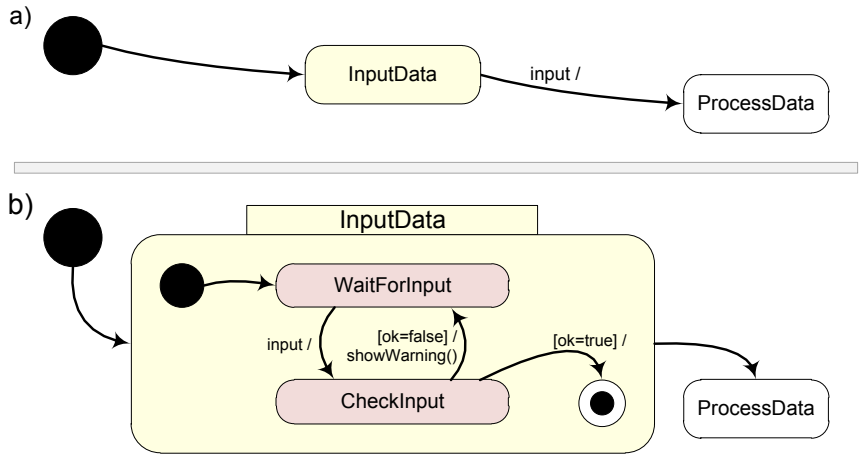

Figure 2.4: Example statechart for data input, a) showing the higher abstraction level, and b) showing a state hierarchy that details the input validation

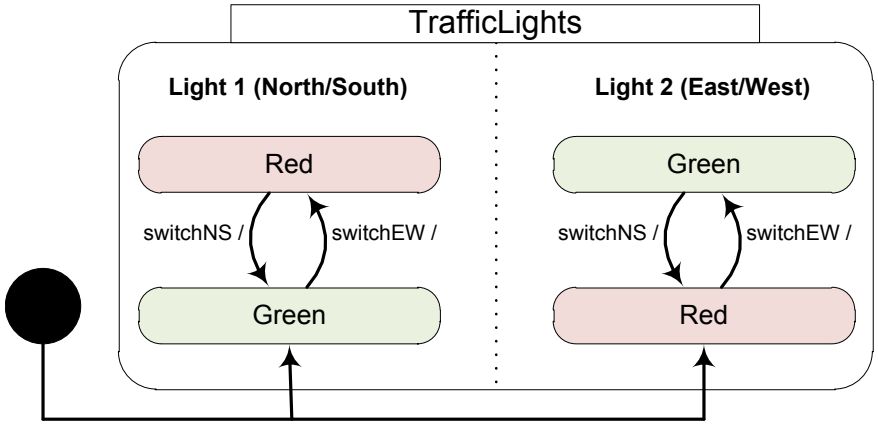

Figure 2.5: Example statechart for two (simplified) traffic lights, showing parallel but synchronized processes in an AND-state

2.4 Use Case

2.4.1 Mobile Telecommunications

The concepts developed in this dissertation are primarily applied to the mobile telecommunications domain, specifically to recommend appropriate mobile phones to consumers. Backed by a research project conducted in cooperation with the :a:k:t: Informationssysteme AG[3], a local industry partner, the implementation for this use case relies on real data, models, and sales knowledge, demonstrating the applicability of the approach to a real-life scenario with industry-strength requirements.

Some properties of the mobile telecommunications domain make it particularly suitable for using electronic recommender systems:

- *The products are relatively complex.* Essentially, today's mobile phones are multipurpose devices with a variety of technical functions and complicated properties that have to be taken into account when customers want to make an optimal buying decision. Also, customers would generally not immerse themselves sufficiently into these technical details to obtain the necessary knowledge individually. It can therefore be assumed that customers have to rely on informed external advice when choosing a mobile phone and that, consequently, vendors who offer qualified consultation to their prospective customers will enjoy the advantage of increased customer satisfaction. A recommender system can provide this service in fields where an individualised consultation would normally be infeasible, e.g., in discount shops or for online / WWW-based scenarios.
- *The domain changes frquently.* Driven by a constant need of innovation to remain competitive in the market, manufacturers frequently release new products, often with new technical properties that address new customer wishes, up to implying entirely new business models. For example, consider the widespread integration of GPS receivers in mobile phones that led to the development of location-based services and which has a number of additional implications, e.g., in relation to privacy. Hence, even salespersons with profound knowledge of this specific domain find that the level of knowledge ages quickly and they have to spend significant effort to keep themselves up-to-date. A recommender system can be maintained in a centralized way and thus enables more timely and cost-efficient adaptations of the domain knowledge.

[3] `http://www.akt-infosys.de/`

While the complexity and changeability properties are particularly apparent in the telecommunication use case, they are very general and shared by many other domains. For this reason, the developed concepts were used to implement recommenders for a few other applications, in order to demonstrate the transferability of the approach to a wider range of possible use cases.

2.4.2 Movies / DVDs

Having a real-life use case absolutely has its benefits, but debugging the application and analyzing its behaviour based on this use case requires considerable effort. The domain model is very large and, due to its complexity, cannot be sensibly maintained by developers without assistance from domain experts. Ironically, one can see this as a side effect of our design goal to separate the concerns of domain experts and software engineers.

Hence, we decided to model a domain that is slightly smaller and simpler than our primary use case so that we are able to use it more efficiently in experimentation. We have chosen the movies domain, i.e., recommending DVDs, since many people have an intuitive knowledge about the domain based on their own experiences and preferences (unlike the mobile phone domain, where people generally have experiences, but do not often form strong opinions on the domain as a whole).

Our 'reference' movies domain model was created during a workshop by the members of the chair of information management. Despite the low effort involved in creating this second domain model, it produces plausible recommendations when tested with a number of stereotypical movie buyers' personalities. Although no formal validation/evaluation was carried out (to measure, e.g., recommendation quality), the movies domain serves as the proof-of-concept for the transferability of the developed approach.

2.4.3 Courses of Study

Another example domain was built using an early prototype of the recommender system during the Open Day of the University of Passau in 2008. In this instance, we attempted to recommend the courses of study offered by our university to prospective students.

Our domain model contained questions about the school subjects in which the students completed their "Abitur" (german general qualification for university entrance), and was refined with questions about desirable personal traits with the help of the student bodies of our faculties.

The prototype was used by a large number of members of the general public and our evaluations of the experiment led to first insights about the dialogue management, as will be detailed in section 5.2.

3 Metamodel Specification and Semantics

The *domain metamodel* defines the "language" that can be used to describe the domain of discourse of the system, i.e., the market segment that is targeted. The behaviour of the recommender system is defined exclusively in terms of the metamodel, making it independent of the concrete domain at hand. As such, the metamodel, along with its instances, can be seen as the interface between implementers and users of the recommender system, which requires us to make a careful transition between different levels of abstraction, complexity, and formality.

We have chosen to define the metamodel semi-formally as a set of UML class diagrams (cf. [BRJ05]), focussing on the intended concepts and therefore remaining at an abstract level. The core design decisions underlying the metamodel reflect the intent of the recommender system to collect information about customers, namely by asking them questions during an interactive dialogue. From there, the system reasons about other information by exploiting the causal influences between the modelled elements. Figure 3.1 shows the basic structure of the metamodel: information that may be elicited or reasoned about is represented as ReasoningElements (elaborated on in sections 3.1 and 3.2) and the causal Influences are modelled as associations between ReasoningElements (elaborated on in section 3.3).

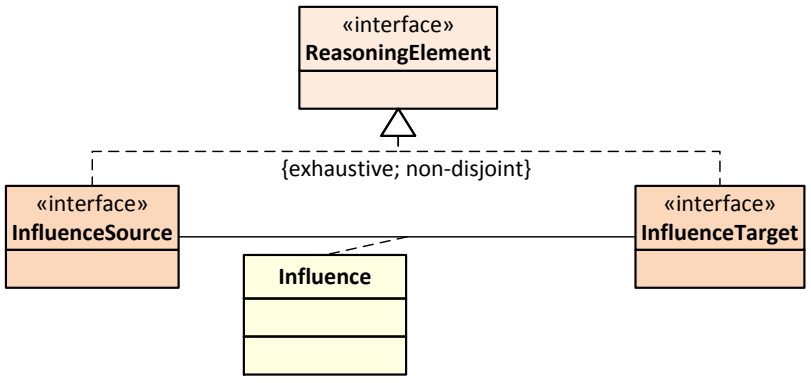

Figure 3.1: Metamodel basic structure: Influences connect ReasoningElements

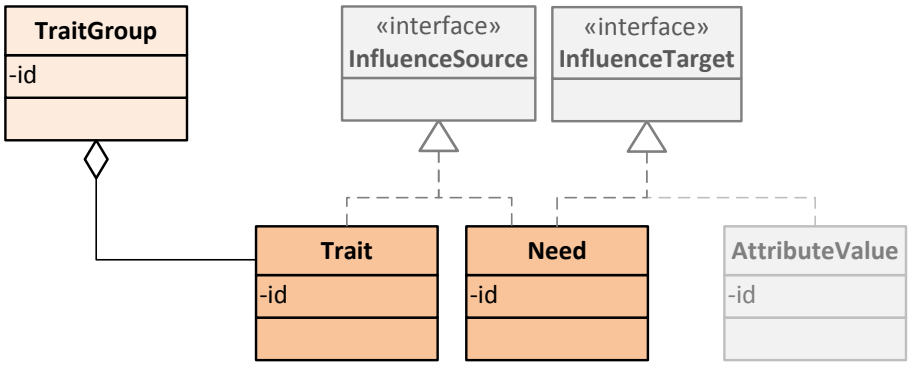

Figure 3.2: UML diagram of the customer metamodel

Modelling the domain of discourse by enumerating the relevant elements and defining the interrelations between them is the central aspect of our approach, as we already presented in [RZF09, RF07]. When analysing the "natural" recommendation processes in the domain of our project, we found such a modelling to adequately reflect the practice of salespersons in the field. Hence, domain experts should be able to express their knowledge easily and intuitively using the means provided by the metamodel (see also section 9.1).

Noteworthy is the implicitness of the modelling, which becomes particularly apparent when comparing our design to other approaches aimed at specifying conversational recommender systems, such as [AFF+03]. Rather than *explicitly* describing a series of dialogue steps, possibly with conditional branches and other similar control elements, we model in a declarative instead of a procedural way only the *knowledge* that is required for the recommendation process. The actual dialogue is then constructed by the recommender system based on inferences from the model. Chapter 5 further elaborates on this concept.

3.1 Customer Metamodel

The *customer metamodel* provides the means to describe information about customers (e.g., personal traits, demographic information, interests, and personal preferences) that the system must consider during the recommendation process. Figure 3.2 shows the customer metamodel on an intentionally abstract level.

In the course of analyzing the way sales experts describe customers, two different concepts emerged which we called Traits and Needs. Both share the idea that the relevant knowledge about the customer is composed of essentially *Boolean* bits of information, following the practice of salespersons to *classify* their customers into a set of categories. When non-Boolean information is requested from a customer, this is typically only done to check whether it falls into a certain category, i.e., the obtained knowledge actually is Boolean again.

- Traits represent facts about a customer, modelled as Boolean properties, representing whether a certain Trait applies to a customer or not. Commonly, Traits are created to represent *classes* of non-Boolean customer properties, e.g., consider a Trait "young" with $(young = true) \iff (age \in [0, 20])$. In other words, the non-Boolean property *age* is converted to a Boolean Trait by classification. Since they are supposed to be objective information, Traits cannot be influenced by other metamodel elements (i.e., they can only function as an InfluenceSource). Furthermore, this objectivity can be exploited, e.g., to pre-answer corresponding questions from a CRM software (if the customer can be identified when using the recommender system).
- Needs are used to describe customer properties beyond standard personal information, such as opinions, preferences, and demands. Therefore, when eliciting information about Needs, the typical answer will not be a binary *yes* or *no* but instead a more fine-grained representation of the degree of agreement (we will elaborate on this in section 4.1).
  As indicated by the implemented interfaces in fig. 3.2, Needs can be influenced by other elements of the metamodel and can, in turn, influence other elements, as we will describe in detail in section 3.3.

The metamodel also provides the option to combine Traits into TraitGroups to support the common case that some Traits are mutually exclusive with each other.

For instance, if the customer's age is modelled as a number of Traits representing different age classes (see example 3.1.1 below), exactly one of those Traits can apply to him/her and combining the Traits into a suitable TraitGroup allows the recommender system to take advantage of this knowledge.

Creating concrete domain models is accomplished by *instantiating* the classes provided by the metamodel as illustrated by the following example.

Example 3.1.1 (Customer model for the mobile phone domain) Let us consider a simplified customer model for selling mobile phones that is built on a single TraitGroup instance:

Let the only considered customer property be his/her age, modelled by creating the TraitGroup "ageclasses", separated into ranges by the conditions shown in table 3.1. Each range is represented as an instance of the class Trait in the customer model and is annotated with its proportion of the population in Germany [Sta10].

Trait	**condition**	**proportion**
$ageclass_{junior}$	$age \leq 20$	20%
$ageclass_{middle}$	$age \in [21, 54]$	49%
$ageclass_{senior}$	$age \geq 55$	31%

Table 3.1: Ranges for TraitGroup "ageclasses"

Further assume that four Needs have been identified for the domain. Therefore, we create four instances of the class Needs. As the recommender system will elicit knowledge about the Needs by engaging in a dialogue with the customer, we illustrate them by giving a sample wording of a corresponding question.

- **Multimedia:** *Does the user intend to use the new mobile phone to create or play back multimedia content?*
- **Music:** *Does the user intend to use the new mobile phone to listen to music?*
- **Office:** *Does the user intend to use the new mobile phone as a mobile office?*
- **Internet:** *Does the user intend to access the internet with the new mobile phone?*

Despite its limited complexity, example 3.1.1 illustrates the central design concepts of the customer metamodel: The customer's age can be ascertained without ambiguity. Based on this information, the corresponding value classes are modelled as Traits. On the other hand, questions regarding the intended usage of the new mobile phone are likely to be answered in a non-Boolean fashion and therefore modelled as Needs. Also, a customer's age is likely to influence the other Needs in the customer model. Additionally, it is plausible that some of the Needs are causally related to each other.

Capturing the causal relationships between Traits and Needs is the essential part of building an appropriate domain model and will be revisited in section 3.3.

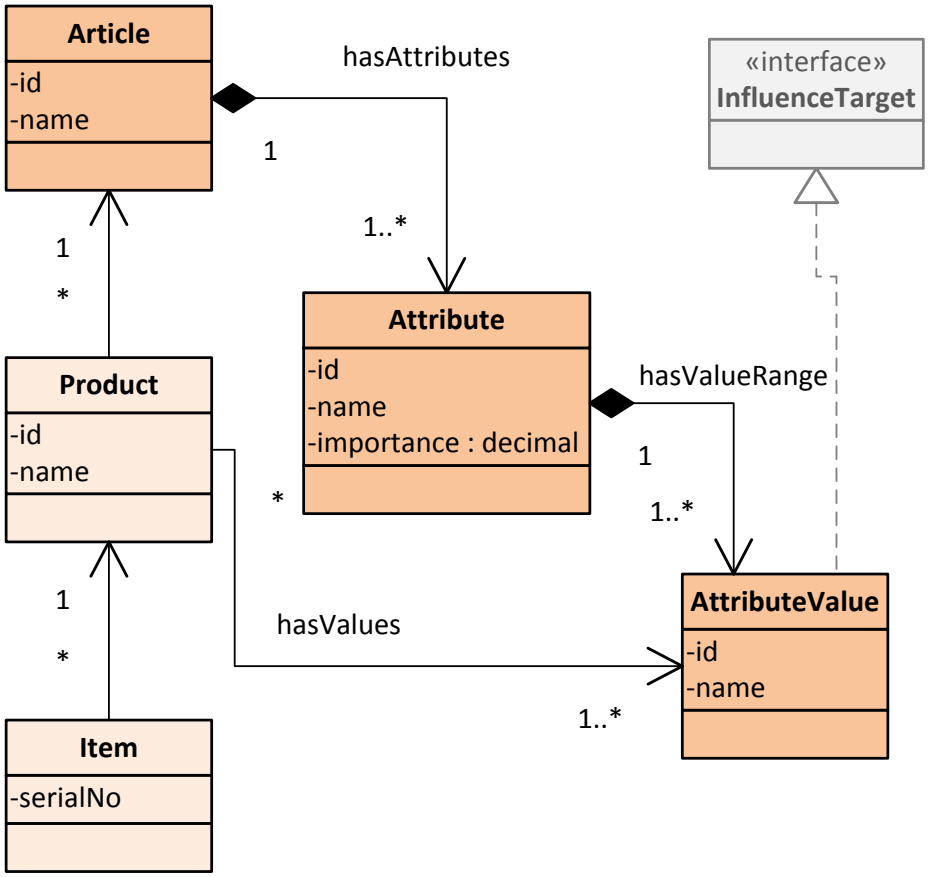

Figure 3.3: UML diagram of the product metamodel

3.2 Product Metamodel

The product metamodel described here allows for modelling the Articles in the domain of discourse and also allows for expressing information about Products.

Individual Items are not represented in the metamodel in an elaborate way, but are left to dedicated order-management or store-keeping systems that come into play when the recommendation process is completed successfully. Items may be relevant for the recommendation process if the recommender system is integrated with these tools.

The product domain is modelled by using a simple composition of parts (see figure 3.3):

- Articles have a finite number of Attributes, which may either be technical (e.g., camera resolution or compatibility with a certain multimedia file format) or non-technical (i.e., sales-oriented, with terms like "chic" or "exclusive").
- Attributes have a finite, explicitly enumerated range of AttributeValues, representing all possible values that an Attribute may have (e.g., for camera resolution, all megapixel values of all cameras in currently available mobile phones). They also have a numeric importance value that denotes the significance of the Attribute for the recommendation process, as will be detailed in section 7.1.
- Products always instantiate a particular Article. The modelling allows specifying all concrete AttributeValues that the Product has.
- Items instantiate a Product and add identifying information, such as serial numbers, storage locations or similar data.

The modelling concepts for products were developed in cooperation with our industry partner and closely follow the naming conventions used by domain experts in the field. Therefore, although all elements are situated on the "metamodel" layer for the recommender system, they represent different abstraction levels in practice and it is important to correctly differentiate between them. Table 3.2 lists the terms used and illustrates them by giving an example for each (cf. also figure 3.4).

concept	**example**
Article	"Mobile phones" in their *entirety*
Product	"Nokia N9" mobile phone *series*
Item	*concrete* Nokia N9 phone with IMEI #357923042078495

Table 3.2: Abstraction levels for the product modelling

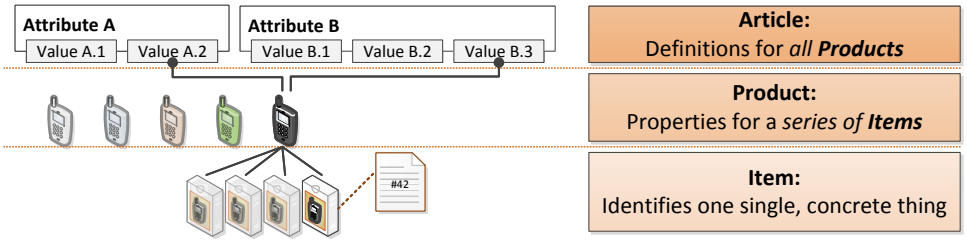

Figure 3.4: Product modelling abstraction levels illustrated

In analogy to the customer metamodel, concrete product models are created by instantiating the appropriate classes of the product metamodel, as the following example illustrates.

Example 3.2.1 (Product model for the mobile phone domain) Let us build on example 3.1.1 by considering a grossly simplified product model for the Article "mobile phone":

Let the mobile phone have three Attributes with AttributeValues and importance values corresponding to table 3.3. So, as a whole, our product model consists of one instance of the class Article, three instances of Attribute and seven instances of AttributeValue, linked by the appropriate composition relations according to figure 3.3. We do not model instances of the Product or Item classes in this example.

Since Attributes with only two AttributeValues can very often be interpreted as Boolean variables, denoting the presence or absence of a particular capability, alternative values of "*true*" and "*false*" are given where appropriate.

Attribute (importance)	AttributeValue
MP3 playback (2)	available (*true*) not available (*false*)
Internal memory size (1)	small (e.g., *<1GB*) medium (e.g., *1–8GB*) large (e.g., *>8GB*)
UMTS/3G ability (2)	capable (*true*) incapable (*false*)

Table 3.3: Attributes and AttributeValues

The example also illustrates the way to limit the value ranges of Attributes to finite sets, i.e., in this case, by dividing various memory sizes into three discrete *categories* (as given in table 3.1.1).

Note that a discretisation step like this implicitly allows to further develop the product model, e.g., by redefining the meaning of "large memory" based on technological improvements without having to touch the modelled Attributes and AttributeValues themselves.

3.3 Interrelations

Up until now, we have defined the product and customer metamodels in an isolated way, without fully elaborating on the possible relationships between the modelling elements. Examples 3.1.1 and 3.2.1, however, already suggested obvious causal dependencies between some elements. For example, the customer's age is very likely to influence his/her multimedia affinity and office requirements, which, in turn, will imply certain technical requirements. The picture will be completed in example 3.3.6 below, after we have explained the concept of an Influence as shown in figure 3.5.

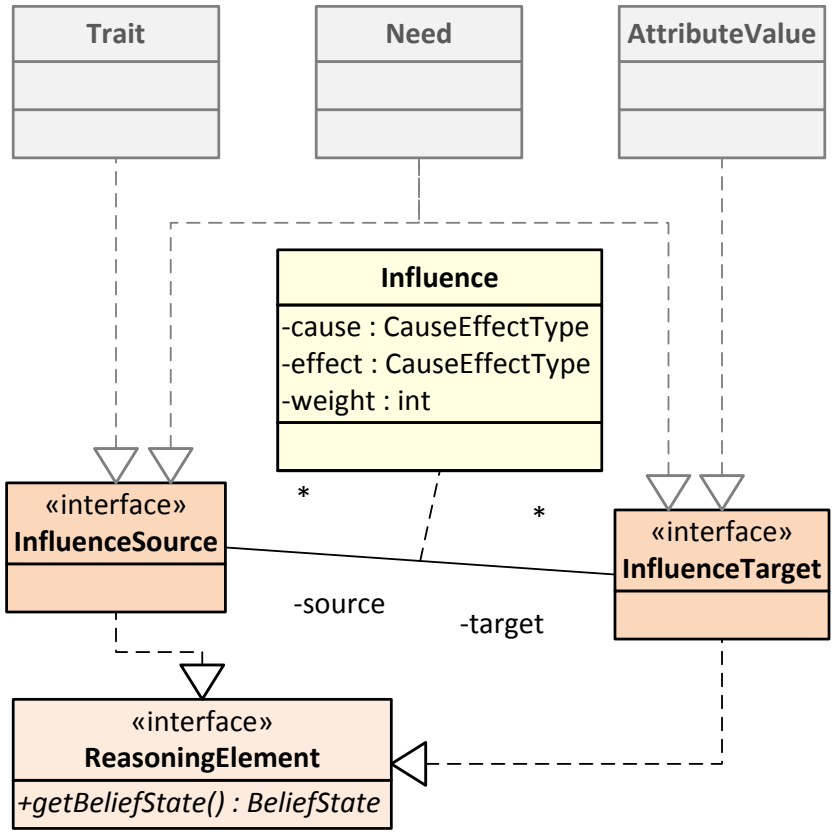

Figure 3.5: UML diagram of the interrelationships metamodel

As already shown in figure 3.1, interfaces separate ReasoningElements into categories according to whether they may *have an influence* on some other modelling elements (InfluenceSource, e.g., Trait), or whether they *can be influenced* by other modelling

elements (InfluenceTarget, e.g., AttributeValue). It is possible that modelling elements both influence other elements and can themselves be influenced (in fact, this is exactly the case for the class Need, which simply implements both interfaces).

Given this separation, Influences are defined as follows:

Definition 3.3.1 (Influence) An Influence represents a potential causal dependency between one element of type InfluenceSource with one InfluenceTarget element. The relationship is further characterised by a *cause* and an *effect*, both being of type CauseEffectType which is an enumeration of "positive" and "negative". It also has a numerical *weight* to signify the strength of the Influence, i.e., its ability to modify the BeliefState of its *target* given appropriate knowledge available for its *source* (see definitions 3.3.2 and 3.3.4 below).

Effectively, Influences can be regarded as the edges of a directed graph whose nodes can be separated into Traits that are only sources, AttributeValues that are only sinks and Needs that may function as both sources and sinks at the same time. Since Influences are intended to model a "causal" relationship, the graph is required to be *acyclical*.

To explain the semantics of Influences more clearly, we will give a definition of the BeliefState that can be determined for every ReasoningElement in an instance of the metamodel. The concept itself is largely analogous to the corresponding notion introduced in [RN10, section 4.4], but in our conceptualization there exists a separate BeliefState for every ReasoningElement, as opposed to [RN10] where the belief state is only a single parameter of the whole system that is being observed.

Every ReasoningElement may be in one of two states, namely *positive* and *negative*. The particular meaning of those states depends on the type of the ReasoningElement:

- For Traits, the BeliefState signifies whether the particular Trait applies to the customer that the recommender system is reasoning about.
- For Needs, the BeliefState signifies whether the customer feels the particular Need.
- For AttributeValues, the BeliefState signifies whether a Product with that particular AttributeValue will satisfy the customer's wishes (in relation to the corresponding Attribute).

Since the inference algorithms will employ techniques for uncertain reasoning, the system's beliefs cannot be adequately represented by a Boolean value. Therefore, the

BeliefState uses a *degree* value, to model its *confidence* that the ReasoningElement is in the positive state. The degree is frequently represented as a number in $[0, 1]$ and it is implied that a complementary degree of belief exists regarding the negative state of the ReasoningElement.

These considerations allow us to define the BeliefState as follows:

Definition 3.3.2 (Belief State) Given a ReasoningElement r, the BeliefState β_r of r represents the system's belief about the state r is in, given all knowledge available to the system.

$\beta_r = \{(positive, \beta_r^+), (negative, \beta_r^-)\}$ with *positive* and *negative* the states that r may be in and $\beta_r^+, \beta_r^- \in [0, 1]$ the corresponding *degrees of belief* (i.e., a measure of the confidence that the system is in the corresponding state, given all knowledge available to the system).

Example 3.3.3 (Belief State) Given a certain state of the recommendation dialogue, the system has determined the following belief states for some of the ReasoningElements in its customer model (cf. example 3.1.1):

$\beta_{Multimedia} = \{(positive, 0.7), (negative, 0.3)\}$, meaning that the customer will more likely have a need for multimedia functions than not.

$\beta_{Music} = \{(positive, 0.5), (negative, 0.5)\}$, meaning that no clear statement about the customer's music requirements can be made at this point of the dialogue.

$\beta_{Office} = \{(positive, 1.0), (negative, 0.0)\}$, meaning that the system is certain that the customer will need office functions.

$\beta_{Internet} = \{(positive, 0.15), (negative, 0.1)\}$, meaning that the system's predicition confidence regarding the customers internet need is very low, albeit with a slight tendency towards the *positive* state.

Note that a degree of belief is *not* directly equivalent to a likelihood, but merely a numeric representation of a confidence estimation normalized to $[0, 1]$. However, since we will use probability theory anyhow later in our implementation, equalling a degree of belief to a probability may serve as a useful approach to make the abstract concept of belief states more intuitive. A key difference is that definition 3.3.2 does not mandate a dependency between β_r^+ and β_r^- as would be the case for probabilities (where β_r^+ and β_r^- would always sum to 1.0), which is illustrated by the last point in example 3.3.3.

Using the concept of BeliefStates introduced above, a definition of the semantics of the cause-effect relationships represented by Influences can be given:

Definition 3.3.4 (Semantics of an Influence) The semantics of an Influence i is defined as follows:

- If the system's degree of belief in the *cause* of i is modified (e.g., by elicitating appropriate knowledge from the customer), the degree of belief in the *effect* is modified appropriately as well, taking the weight of the Influence into account (the exact way of doing this will be defined in chapter 4).
- In all other cases, the degree of belief for the effect remains unchanged. (Note that, depending on the concrete implementation, a change may result from the fact that an increase in confidence for the source has become impossible.)

It is noteworthy that an Influence only models *direct* causal dependencies between two ReasoningElements. The concept of a single Influence does not encompass "chains" or network-like structures of causal interrelations. If domains contain such relationships between their modelling elements, a correspondingly great number of Influences will need to be created.

Example 3.3.5 (Semantics of an Influence) Based on example 3.1.1, consider two Needs, *multimedia* and *music*, and let i be an Influence with $i.source = multimedia$, $i.target = music$, $i.cause = positive$, $i.effect = positive$, $i.weight = 2$. In other words, the Influence models that customers who want multimedia functions will likely also want music playback functions.

Now, if a given dialogue interaction causes the recommender system to modify its estimation for $\beta^+_{multimedia}$, e.g., increasing it from 0.5 to 0.7, it also has to update its estimation for β^+_{music} by increasing the value. Let us assume for now that the increase for the effect is implemented as being directly proportional to the change of the cause, so β^+_{music} is also increased from 0.5 to 0.7 as a result from evaluating i.

Example 3.3.6 (Interrelations model for the mobile phone domain) Let us complete examples 3.1.1 and 3.2.1 by providing a number of causal interrelations between the elements as described here:

- Young customers (i.e., aged 21 and younger) will have a demand for multimedia functions.
- Customers with multimedia needs will...

  - ...also have a need for music-specific functions;
  - ...prefer devices with a large internal memory;
  - ...also settle for devices with a medium amount of internal memory if necessary.
- Customers with a need for music functions will request devices with MP3 players while finding devices without one inacceptable.
- Adult customers (i.e., aged between 22 and 54) will have a demand for office functions.
- Customers with office needs will...
  - ...also need internet functions;
  - ...find too small amounts of internal memory inacceptable;
  - ...be satisfied by a medium amount of internal memory;
  - ...not gain a substantial advantage from very large memory sizes.
- Customers with a demand for internet functions will request devices with broadband wireless connectivity such as UMTS and deem devices without those capabilities inacceptable.

These informal descriptions for causal dependencies are summarised in table 3.4 which lists the instances of Influence that would need to be created. It is noteworthy that the *weights* are strictly limited to the values 1 and 2, respectively. We will provide an explanation for this in section 3.4 below.

Note that all Influences in example 3.3.6 have *positive causes*. Apart from simplifying this concrete example, we have found very few instances where an Influence with a negative cause was created, leading to the conclusion that, at least in our application project, salespersons do not frequently draw conclusions from the *absence* of Needs for a customer. The only examples of Influences with negative causes where generally associated with Needs relating to customers' price-sensitivity.

Example 3.3.7 (Running Example) Figure 3.6 combines examples 3.1.1, 3.2.1, and 3.3.6. This combined model of the mobile phone domain will serve as the running eaxmple for the remainder of this thesis.

source	target	cause	effect	weight
age $\leq$ 21	multimedia	positive	positive	2
22 $\leq$ age $\leq$ 54	office	positive	positive	2
multimedia	music	positive	positive	2
multimedia	memory: large	positive	positive	2
multimedia	memory: medium	positive	positive	1
music	MP3: available	positive	positive	2
music	MP3: not available	positive	negative	2
office	internet	positive	positive	2
office	memory: small	positive	negative	2
office	memory: medium	positive	positive	1
internet	UMTS: capable	positive	positive	2
internet	UMTS: incapable	positive	negative	2

Table 3.4: Sample Influences

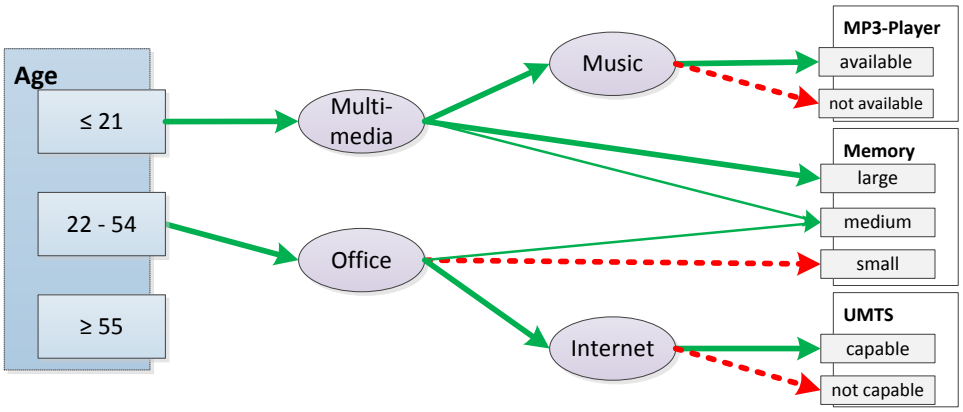

Figure 3.6: Sample model for the mobile phone domain

Solid green arrows denote Influences with a *positive* effect, whereas dotted red arrows denote Influences with a *negative* effect (as mentioned, all causes are positive). The line thickness of the arrows signifies the *weight* of the corresponding Influences (i.e., 1 or 2).

In order to illustrate the way how ReasoningElements interact with Influences, we provide another example that may represent the first step of an actual recommendation dialogue:

Example 3.3.8 (Inference in the example domain) Consider the mobile phone domain as modeled in example 3.3.7 with all its belief states in their initial configurations (e.g., equal degrees of belief for *positive* and *negative* or, alternatively, initial estimations based on demographic sources or similar).

We will now demonstrate how elicited knowledge propagates through the domain model. Since the quantitave changes in the degrees of belief are based on the concrete implementation that is introduced in later chapters, we will keep our example on a mainly qualitative level.

Assume that, as its first question, the system asks about the customer's age and learns that he/she is 20 years old. In order to represent this information in its internal model, the system would update its belief state $\beta_{age \leq 21}$ by increasing the value of $\beta^{+}_{age \leq 21}$ and decreasing the value of $\beta^{-}_{age \leq 21}$. A plausible new belief state would be $\beta_{age \leq 21} = \{(positive, 1.0), (negative, 0.0)\}$.

Following the defined Influences, this leads to changes for the belief states of other ReasoningElements (cf. figure 3.7):

$\beta^{+}_{multimedia}$ must be increased, "transitively" leading to an increase of β^{+}_{music}. These changes, consequently, lead to increases in the degrees of belief for the following AttributeValues:
$\beta^{+}_{MP3\_available}$
$\beta^{-}_{MP3\_not\_available}$
$\beta^{+}_{memory\_large}$
$\beta^{+}_{memory\_medium}$ (to a lesser extent because of an Influece with a smaller weight)

Hence, from the answered question, the recommender system can infer an increased interest in multimedia and music functions, leading to first insights about a desired MP3 player and suitable sizes for the memory size of the mobile phone.

Finally, the system could also adjust its belief states for the other age groups in the domain model (the customer cannot be in any of them), possibly leading to further changes similar to those described above.

3.4 Project Experiences

Let us extend the modelling concepts introduced in the preceding sections by providing some lessons that we learned during the concrete application of the concepts together with our industry partner.

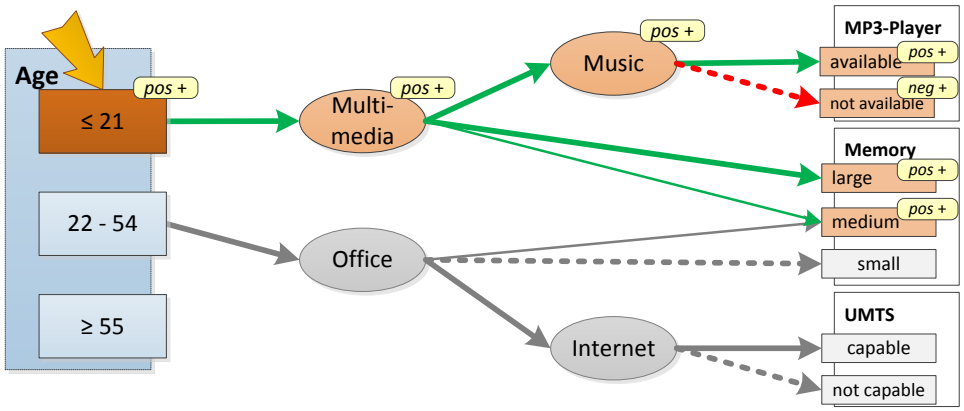

Figure 3.7: Belief state change propagation

Lacking a concrete use-case, we did not explore multi-domain recommendation processes in detail, but while instances of the customer metamodel are always built with a concrete market domain in mind, multi-domain models can be created in principle by unifying different customer models and merging synonymous elements. In such a case, knowledge about the customer learnt from the sales dialogue for one domain would then automatically and transparently be reused for recommendations in the other domains.

Analyzing the similarity-based merging of models is beyond the scope of this dissertation and the interested reader is therefore referred to works about ontology matching or ontology merging, respectively (e.g., [Tay10, KS03, CGL01]). The research described in [Hel11] might be of interest, too.

Attributes and their value ranges deserve some further notes regarding our practical experiences with their usage:

- The limitation to finite, explicitly enumerated value ranges may seem overly restrictive at first. As it turned out in practice, the issue was not significant:
  - Many value ranges that would be infinite (e.g., numbers used as weights or sizes) can be discretised in our use-case, since losing some precision often does not matter for the recommendation process. For example, it is not practically relevant to differentiate the weight of mobile phones by single grams. As another example, products are usually priced at important psychological boundaries (e.g., EUR 9.99 instead of EUR 10.35), creating an

implicit descretisation (again, minor variances can be ignored safely).

  - Surprisingly, in the mobile phone use case, the Attribute "colour" proved to be the most difficult Attribute to represent suitably. Since practically all manufacturers use fancy marketing terms to describe their devices' colors (e.g., "steel grey", "midnight blue", or "fiery red") that might or might not represent actual visual differences, the colours could not be adequately grouped together without looking at the concrete items.

- Many Attributes have "Boolean" value ranges, denoting the presence or absence of a particular feature. User interfaces may decide to group related Attributes into lists for display (e.g., a list of supported audio file formats would be modelled as several Boolean Attributes internally).

- It may be necessary to include an AttributeValue with a semantic of "unknown" with every Attribute if the data that can be obtained about products is not always 100% complete. This is particularly often the case for newly published products, according to our project experience, but cannot generally be ruled out for any product. However, the treatment of an AttributeValue with an "unknown" semantic during the recommendation process is generally difficult and dependent on the domain (i.e., should it be treated equal to a detrimental value or neutrally compared to other values?). Removing these values as far as possible is therefore desirable to eliminate negative effects on recommendation quality.

Definition 3.3.1 allows the *weight* of an Influence to be an arbitrary real number. While this is conceptually true, we found it useful to limit the interval from which *weights* can be chosen, because of the way that they will be used later (cf. definition 4.2.4). Furthermore, we wanted to explicitly avoid that domain experts would draw conclusions from the numerical values. Therefore, we decided to further simplify the available modelling options by limiting the available *weights* for Influences to a "strong" and a "weak" Influence in our practical implementation. These are internally represented by the numerical values 1 and 2, respectively, as was already shown in example 3.3.6.

Example 3.4.1 (Real Domain Model Sizes) Example 3.3.7 serves well to demonstrate the significant points of our approach. In order to illustrate the size of our real world use case, let us give another example. The actual mobile phone sales domain model of our prototype consists of:

- A single TraitGroup called "customer stereotype" with the following Traits:
  - Business customer

  - Fun-oriented user
  - "Gearhead"
  - Craftsman (quite surprisingly for the outsider, craftsmen actually are a specific marketing target group for mobile phone sales, preferring robust, simple phones with fixed-cost rate plans)
  - Individual customer (sort of an "anything else" answer option)

  By grouping these Traits into a TraitGroup, it is implicitly assumed that a customer may be classified into exactly *one* of these stereotypes.

- 22 Needs, such as "Multimedia", "Price sensitive", "Gaming", or "Comfort".
- 78 Attributes with a total of 374 AttributeValues.
  - Of these, 51 are Attributes with a Boolean value range, denoting the presence or absence of a particular feature.
  - All value ranges contain a dedicated *"NONE"* AttributeValue to model lack of knowledge explicitly, i.e., even "Boolean" value ranges contain 3 elements.
  - The modelled Attributes range from purely technical properties such as "weight", "camera resolution", or "manufacturer" to artificial marketing terms like "target audience", demonstrating the flexibility of the modelling.
- 1198 Influences, resulting in a maximum parent degree of 10 for a ReasoningElement (we anticipate that the processing of a domain model has a complexity that is exponential with respect to the parent degree).

To give a rough idea of the size of the domain model, fig. 3.8 shows a rendering of the graph defined by the ReasoningElements and Influences (omitting non-connected ReasoningElements). Generally, Traits are located in the center of the image, ringed by Needs, while AttributeValues would form the "leaves" on the outsides.

3.5 Conclusion

In this chapter, we presented a metamodel for modelling domains for recommender systems. The metamodel represents the first main contribution of this dissertation. It can be separated into a customer metamodel and a product metamodel.

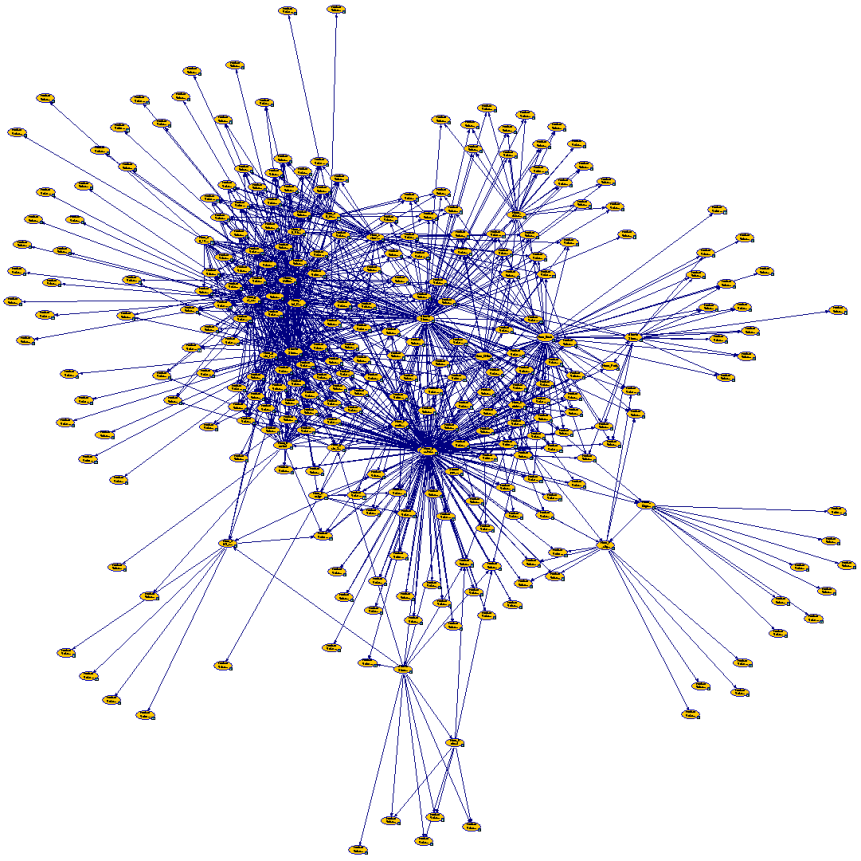

Figure 3.8: Sample rendering of the real domain model

The customer metamodel allows for specifying customer properties such as age, profession, or similar objective information. Furthermore, it provides means for modelling expectations, desires and needs that customers have regarding the product they intend to buy. Elements from this model will later be used by the recommender system to elicit the appropriate information from concrete customers by engaging in a dialogue with them (cf. chapter 5).

The product metamodel, on the other hand, allows for modelling the technical properties of the products that are sold in the domain. For each technical attribute, a finite range of possible values has to be specified. Instances of the product metamodel will later be used by the recommender system to derive a utility function that is used to rank the contents of the product catalogue according to the customer's wishes (cf. chapter 7).

Elements from both parts are connected by Influences that represent the causal interrelations that may exist between customer needs and the technical properties that fulfill these needs. The Influences are an essential component in generating the inference engine that forms the core of our recommender system (cf. chapters 4 & 6).

4 Metamodel Implementation

4.1 Intermediate Representation

4.1.1 Intermediate Model

The intermediate model is designed as a connective link between the more intuitive, use-case oriented domain metamodel and the concrete implementation of the inference engine.

As such, it must be formal and concise enough to allow for a well-defined implementation of its intended semantics. In fact, we are able to prove the soundness and completeness of our concrete implementation with respect to the intermediate model in section 4.2.2. Secondly, it must be expressive enough to support the representation of the behaviour of the recommender system as implied by instances of the domain metamodel.

Definition 4.1.1 (Intermediate Model) The *intermediate model* $\mathbf{I} = (\mathbf{V}, \mathbf{E})$ is defined as follows:

$\mathbf{V}$ denotes a set of nodes representing *variables*. Every $V \in \mathbf{V}$ has an associated discrete finite *value range* $dom(V)$ and a *BeliefState* β_V (cf. definition 4.1.2 below). There are two sets $\mathbf{S}$ and $\mathbf{T}$, not necessarily disjoint, with $\mathbf{V} = \mathbf{S} \cup \mathbf{T}$. $\mathbf{S}$ denotes the set of *sources* and $\mathbf{T}$ denotes the set of *targets*.

$\mathbf{E}$ denotes the set of *edges* E with $E = (S \in \mathbf{S}, s \in dom(S), T \in \mathbf{T}, t \in dom(T), w \in \mathbb{R})$ where S is the *source variable* of E, s is the *source value* of E, T is the *target variable* of E, t is the *target value* of T, and w is the *weight* of E.

Given $E = (S, s, T, t, w)$, we often call the tuple (S, s) a *cause* and the tuple (T, t) an *effect* with respect to E.

Before describing the process to obtain an intermediate model from a domain model in detail in section 4.1.2, we will provide a definition of the semantics of an edge in

the intermediate model which will be closely along the lines of definition 3.3.4.

Definition 4.1.2 (Belief State in the Intermediate Model) Given an intermediate model $\mathbf{I} = (\mathbf{V}, \mathbf{E})$, the *Belief State* β_V of a variable $V \in \mathbf{V}$ is defined as follows:

$\beta_V = \bigcup_{v \in dom(V)} \{(v, \beta_v)\}$ where we call β_v the system's *degree of belief* relating to the value v with $\forall (v, \beta_v) \in \beta_V : \beta_v \in [0..1]$.

Note that probability theory would give us an obvious implementation of the belief state as given by definition 4.1.2. However, at this point of our work, we do not want to limit ourselves to regard a degree of belief exclusively as a probability of the corresponding event. Restricting intermediate models to such a narrow view would not provide significant advantages, while, on the other hand, it would possibly preclude alternative approaches to be described in terms of the intermediate model.

For example, if using probability theory, a restriction of $\sum_{v \in dom(V)} \beta_v = 1.0$ would have to be observed by all implementations, which is intentionally not enforced by definition 4.1.2 to allow a greater freedom of implementation.

We could envision, e.g., implementations based on fuzzy logic, pure propositional logic, or even entirely different ways of reasoning. As long as they are able to express their internal inference results in terms of a belief state as introduced by definition 4.1.2 (subject to the constraints imposed by definition 4.1.3 below), these approaches could be subsumed by our modelling.

Nevertheless, we agree that probability theory is a very suitable way of reasoning about belief states. After all, the concept was also chosen as the basis for the implementation of our own inference engine (see section 4.2).

Definition 4.1.3 (Semantics of an Edge in the Intermediate Model) Let $E = (S, s, T, t, w)$ be an edge in an intermediate model $\mathbf{I} = (\mathbf{V}, \mathbf{E})$. The *semantics of* E then define how changes in the belief state of the source variable S propagate to the target variable T.

Assume that, due to newly elicited knowledge, the system's belief state for S changes from $\beta_S = \{\ldots, (s, \beta_s), \ldots\}$ to $\hat{\beta}_S = \{\ldots, (s, \hat{\beta}_s), \ldots\}$.

Then, the existence of E implies corresponding changes in β_T, as follows:

- If $\hat{\beta}_s \geq \beta_s$, it holds that $\hat{\beta}_t \geq \beta_t$.
- If $\hat{\beta}_s \leq \beta_s$, it holds that $\hat{\beta}_t \leq \beta_t$.

The weight w controls the strength of the propagation, i.e., the larger the value w, the larger will the value of $|\hat{\beta}_t - \beta_t|$ be (for a given $|\hat{\beta}_s - \beta_s|$ and all other things being equal). The exact numerical meaning of w is left to the implementation, however.

This definition is largely analogous to the semantics of Influences in definition 3.3.4. Whereas Influences allow to express the system's beliefs only in respect to complete ReasoningElements, the intermediate model provides a more fine-grained specification of the change behaviour that is useful for concrete implementations.

Given an intermediate model, a concrete implementation only needs to define how its internal workings relate to the concepts of "degree of belief" and how exactly the "weight" factor is to be interpreted. For our concrete implementation, we will answer these questions in section 4.2.

4.1.2 Generation from Domain Model

The procedure to generate an intermediate representation $\mathbf{I} = (\mathbf{V}, \mathbf{E})$ from a given domain model is straight forward. First, the ReasoningElements have to be appropriately converted into variables. Then, the Influences have to be converted into edges.

4.1.2.1 Converting ReasoningElements

When transforming domain models into their respective intermediate representations, one variable $V \in \mathbf{V}$ is created for every ReasoningElement. More precisely, Traits and Needs (i.e., all implementors of the InfluenceSource interface) are considered sources $\in \mathbf{S}$, whereas Needs and AttributeValues (i.e., InfluenceTarget's implementors) are targets $\in \mathbf{T}$. Note that $\mathbf{S} \cap \mathbf{T}$ is generally not empty, since it likely contains Needs (cf. fig. 3.2).

A value range must be defined for all created variables. Along the lines of definition 3.3.2, Traits and AttributeValues receive simple binary value ranges since either their positive or their negative state can be applicable in a particular situation. Therefore, for a variable V that was created from a Trait or an AttributeValue, $dom(V) = \{pos_1, neg_1\}$.

For Needs, on the other hand, both the positive and the negative state of definition 3.3.2 may be applicable in varying degrees as introduced in section 3.1. To represent this, the generation process requires the specification of a parameter $d \in \mathbb{N}$ (for "detail").

Using d, the value range of a variable V that was created from a Need is defined as $dom(V) = \{pos_i | i \leq d\} \cup \{neg_i | i \leq d\} \cup \{neutral\}$, leading to $|dom(V)| = 2d + 1$.

Effectively, this allows us to represent the degree to which a Need applies on the Likert-scale [Lik32] that serves as a widely accepted standard in questionnaires that require answers to attitude-related questions. In our main use case, we chose $d = 2$ globally for all Needs, leading to value ranges as exemplified in table 4.1.

$v \in dom(V)$	Intuition
pos_2	"strongly agree"
pos_1	"partly agree"
$neutral$	"neither agree nor disagree"
neg_1	"partly disagree"
neg_2	"strongly disagree"

Table 4.1: Possible value range for a Need

Instead of a global parameter, it would also be possible to define d on a per-Need basis by a trivial extension of the domain metamodel (see section 3.1). However, it turned out that domain modellers found it difficult to decide on a specific level of detail for individual Needs, so that we opted for one global value that conforms to the expectations posed to Likert-scales.

Note on Implementation It should be noted that, while these things would be conceptually independent, the cardinality of the value ranges for Need-variables has a direct impact on the performance of *our* concrete implementation. Hence, a modeller must observe the trade-off between a desire for very fine-grained inference and the timing constraints for interactive system usage. Also, very detailed reasoning requires the customers to give equally detailed answers, which may not be appropriate. The default value range has shown to be reasonable in our use cases with respect to granularity, complexity for customers, and performance.

4.1.2.2 Converting Influences

Influences are each converted into a number of edges, which serve to connect the values of the corresponding source variable of the Influence to the appropriate values of the corresponding target variable. Therefore, given a globally fixed parameter d, d^2 edges are created per Influence between Needs.

For example, given an Influence from Need S to Need T with ($cause = positive$, $effect = negative$, $weight = 2$), the following edges would be created if we assume $d = 2$:

$E_{1,1} = (S, pos_1, T, neg_1, w_{1,1})$
$E_{1,2} = (S, pos_1, T, neg_2, w_{1,2})$
$E_{2,1} = (S, pos_2, T, neg_1, w_{2,1})$
$E_{2,2} = (S, pos_2, T, neg_2, w_{2,2})$

The weights are calculated based on the assumption that the more extreme values (i.e., those with a higher index, such as pos_2) represent the clearest statements according to the Likert-scale that the values represent. The weight should reflect this by being higher for edges to and from values with higher indices. Therefore, the indices are factored into the calculation of a weight for an edge as follows.

Definition 4.1.4 (Weight of an Edge) Given an edge $E = (S, val_i, T, val_j, w)$ and i_{max} and j_{max} the maximum possible indices of $dom(S)$ and $dom(T)$, respectively, we define the weight w of E as

$w = \frac{i*j}{i_{max}*j_{max}} * w_{Influence}$

with $w_{Influence}$ being the weight of the original Influence that E was created from,

$$i_{max} = \begin{cases} d & \text{if } S \text{ is a Need} \\ 1 & \text{otherwise} \end{cases},$$

and, analogously,

$$j_{max} = \begin{cases} d & \text{if } T \text{ is a Need} \\ 1 & \text{otherwise} \end{cases}.$$

Example 4.1.5 (Weight of an Edge) As an example, let $E_{1,1}$, $E_{1,2}$, $E_{2,1}$, and $E_{2,2}$ be as above. With $w_{Influence} = 2$, we get $w_{1,1} = \frac{1}{2}$, $w_{1,2} = w_{2,1} = 1$, and $w_{2,2} = 2$ for their respective weights, since $i_{max} = j_{max} = d = 2$.

Note that there are no edges generated to and from the *neutral* values that may be in the value range of variables that correspond to Needs when using the technique as described. Both in our primary use case and in all other domains that were examined, we never found a requirement to support drawing conclusions from customer statements that would correspond to *"I don't know."* or *"I don't care."* statements or, vice versa, causal relationships that would explicitly lead the recommender system's belief towards one of these statements for a ReasoningElement.

A Note on Generality Of course, the absence of such a requirement cannot be proven for all imaginable application domains. Hence, should such a requirement be found, the domain metamodel would have to be extended – specifically, a way to accomplish this would be to allow more values for the CauseEffectType as shown in figure 3.5. Given such an extension, the process being described in the current section would have to be expanded as well. However, instances of such an extension would be within the definition of the intermediate representation and its usage as described in section 4.2 would not be affected. Having to implement only local changes in spite of a significant modification is one of the advantages of the layered nature of our approach. We will revisit this topic in chapter 8.

4.2 Implementation Based on Bayesian Networks

4.2.1 Introduction

Bayesian networks as defined in section 2.2 have a number of properties that make them suitable as the underlying formalism for the implementation of the inference engine:

- Bayes networks are designed to represent exactly these kinds of cause-effect relationships that form the core of both the domain model and its intermediate representation.
- Bayes networks are able to represent the uncertainty that is inherently present when reasoning about human emotions and preferences.
- Bayes networks can cope with the incomplete knowledge that must be expected in dialogue management:
  - Since the dialogue flow is very flexible, answers are obtained in an arbitrary order. Bayes networks support this, since they do not place requirements on evidence being inserted in any particular sequence.
  - Some questions might even not be answered at all, which is also supported by Bayesian networks since the inference will transparently use conditional probability distributions for all random variables that do not have associated evidence. As a sidenote, this also means that there is no "minimum amount" of evidence that the system would have to obtain before being able to provide recommendations.

  - Belief revision (i.e., if customers change their opinion during the dialogue) is also supported transparently by simply changing the inserted evidence to represent the corrected answer.

- While our approach primarily uses the *causal* way of reasoning in a Bayes net (i.e., knowledge is perceived as propagating *along* the direction of the edges), the so-called *diagnostic* view of the network may be used to obtain *explanations* of decisions or other system behaviour without further effort.

Furthermore, mature software libraries exist, which are free for education, research, and commercial purposes, to assist the usage of Bayesian networks in practical applications, such as computational biology [FLNP00], information retrieval [dCFLH04], and medicine [DMIZ97].

The intermediate semantics defined in 4.1.1 were therefore implemented by defining an algorithm to transform an instance of the intermediate model into a corresponding Bayesian network. The network has to satisfy the requirement to represent the semantics of the intermediate model as described in definition 4.1.3 in a consistent way, which will be proven in section 4.2.2.

Given an instance of the intermediate model $\mathbf{I} = (\mathbf{V}, \mathbf{E})$, with $\mathbf{V}$ being the set of variables and $\mathbf{E}$ being the set of edges (S, s, T, t, w), we first define the structure of the Bayes net in a straight-forward way:

- Each variable $V \in \mathbf{V}$ is represented in the Bayes net as a single node n_V with a corresponding random variable r_V that has a value range $dom(V)$. To simplify notation, we may also write V for node n_V and random variable r_V, if doing so does not create ambiguity.

- For each edge $E = (S, s, T, t, w) \in \mathbf{E}$ in the intermediate model, an edge (n_S, n_T) and, consequently, a corresponding conditional dependency between the random variables r_S and r_T is added to the Bayes net, unless it is not already present.

  - If $\mathbf{E}$ contains several edges of the form $(s, *, t, *, *)$, only one edge is actually added to the Bayes net. The mutiplicity of edges is then taken into account when constructing the conditional probability tables of the random variables as detailed below.

A central requirement for the applicability of Bayesian networks is the *conditional independence* of the random variables (cf. section 2.2 and [RN10, 13.5] for a general discussion) in the above structure. Namely, each random variable must be conditionally independent of its non-descendants, given its parents. Simply put, if a conditional

dependency between two variables exists, it must be represented in the Bayes net by edges.

We build a so-called "naïve" Bayes model, i.e., conditional indepence between variables is assumed without actually proving it. Such a proof, if possible at all, would have to be done for every targeted domain separately and would require enormous amounts of statistical data. However, naïve Bayesian networks are known to work *"surprisingly well"* [RN10, p.499] (cf. section 2.2.2).

Also, though not a proof, we find it plausible that our Bayes networks are "close" to a truthful representation of the relations between the random variables. This is based on the consideration that modellers *do* have the option of using Influences to express all causal interrelations between variables if they deem them relevant. Doing so would then create a corresponding conditional dependency in the Bayesian network. Consequently, the absence of a dependency implies that, while two given random variables might not be independent, the modeller did not consider the relation between the two corresponding ReasoningElements to be relevant enough to warrant an explicit representation as an Influence.

4.2.2 Bayesian Model

For every random variable generated in the Bayesian network, either a conditional or a plain probability distribution must be specified, depending on whether the respective random variable has any parents.

Definition 4.2.1 (Parents of a Random Variable) Given an instance of the intermediate model $(\mathbf{V}, \mathbf{E})$, let $V \in \mathbf{V}$ be a variable (which has a corresponding node n_V and random variable r_V in the Bayesian network).

The *set of parents* of V is defined as $parents(V) = \{X | X \in \mathbf{V} \wedge \exists((X, *, V, *, *) \in \mathbf{E})\}$.

The *set of parent edges* of V is defined as $parentedges(V) = \{(*, *, V, *, *) \in \mathbf{E}\}$.

First, consider random variables *without parents*. This is always the case if the random variable originates from a Trait in the domain model instance but may also happen for any other variables if the particular domain model instance did not contain corresponding Influences. For these variables, a total probability distribution must be provided. Unless a suitable specific *a-priori* probability distribution can be obtained from an external source (as illustrated, for instance, in table 3.1), a *uniform* probability distribution will be assumed.

More interesting, of course, are random variables that are dependent on other variables. In this case, a conditional probability table ("CPT") must be specified, i.e., *for every possible valuation* of the list of parent variables, a probability distribution has to be provided. Intuitively, the "columns" of the conditional probability table are filled one-by-one depending on their corresponding valuation of the parent variables.

Example 4.2.2 (Sample conditional probablity table) Consider the node/variable "*Memory_medium*" from example 3.3.7. In this case, $parents(Memory\_medium) = \{Multimedia, Office\}$ which means that we have to calculate the conditional probability distribution $\mathbf{P}(Memory\_medium|Multimedia, Office)$.

To save space, we assume that all variables are Boolean and we abbreviate them as M_m, M, and O, respectively. In a tabular form and using the proposition shorthands introduced in definition 2.2.1, the CPT would look as illustrated by table 4.2.

$M = true$		$M = false$	
$O = true$	$O = false$	$O = true$	$O = false$
$P(m\_m \mid m \wedge o)$	$P(m\_m \mid m \wedge \neg o)$	$P(m\_m \mid \neg m \wedge o)$	$P(m\_m \mid \neg m \wedge \neg o)$
$P(\neg m\_m \mid m \wedge o)$	$P(\neg m\_m \mid m \wedge \neg o)$	$P(\neg m\_m \mid \neg m \wedge o)$	$P(\neg m\_m \mid \neg m \wedge \neg o)$

Table 4.2: Sample CPT for *Memory_medium*

We chose a variant of the "Noisy-OR" concept as the general idea behind the construction of our CPTs that we use here in an extended way that works well with the multi-valued random variables of our use case (cf. also [RN10, PPNH94, DG93]).

Generally speaking, Noisy-OR is a more concise way of specifying conditional probability tables that is applicable if the random variables in question follow certain restrictions:

Given an effect and its possible causes, Noisy-OR assumes that each individual cause can trigger the effect *independently* of the other causes (realising the "OR"). On the other hand, each cause may be *inhibited* from triggering the effect (realising the "Noisy" property), again, independently of the other causes. These restrictions make it possible to build conditional probability tables in a systematic way, rather than determining each entry "manually".

Under these independency assumptions, the probability that *several* simultaneous causes do trigger the dependent effect can be determined by looking at the complementary event: If these causes *are* present but *do not* trigger the effect, *all of the causes must have been inhibited*. The probability for this can then be calculated by simply multiplying the individual probabilities that each single cause has been inhibited.

In other words, given a variable *Effect* that is causally dependent on two variables $Cause_1$ and $Cause_2$ as described above, we can calculate $P(effect|cause_1, cause_2)$ as follows (all variables are Boolean and $inhibition(X)$ denotes the inhibition probability assigned to variable X):

$$\begin{aligned} P(effect|cause_1, cause_2) &= 1 - P(\neg effect|cause_1, cause_2) \\ &= 1 - (inhibition(Cause_1) * inhibition(Cause_2)) \end{aligned}$$

Other entries of the conditional probability table can be calculated analogously and the approach scales well to random variables with many parents. In fact, given a random variable variable with n parents, $\mathcal{O}(2^n)$ values would have to be specified to give the full CPT. This is reduced to $\mathcal{O}(n)$ parameters (i.e., the inhibition probabilities of each parent variable) if the causal relationship follows the Noisy-OR restrictions (cf. [RN10, 14.3]).

By introducing a "leak probability", the approach can be extended to allow for the fact that there may be further, unknown causes that may trigger the effect, i.e., allowing for $P(effect|\neg cause_1, \neg cause_2) > 0$.

The central assumptions of our application reflect the restrictions to apply Noisy-OR appropriately, allowing us to use the approach for building our CPTs:

1. Customer needs can be triggered by *multiple causes*. Analogously, a particular technical property may be desirable because of several causes.
2. All these causes can produce the effect *independently of each other*, i.e., every single cause may produce the effect, regardless of other possible causes for the the same effect.
3. However, when reasoning about human preferences and feelings, it is *not certain* that an effect is actually invoked at all, even if one or several of the possible causes are present. Hence, the possible causes must be assigned an inhibition probability to model this fact.

4. For the same reason, on the other hand, the effect may be invoked *despite* the fact that *neither* of the causes are present. I.e., there is always a "base" probability for the effect that can only be increased (by appropriate Influences) – similar to the "leak" probability in the conventional Noisy-OR concept.

Example 4.2.3 (Applicability of Noisy-OR) As an illustration of these assumptions and their applicability to our concrete use case, consider the reasons why people buy Apple phones. These devices are well-known for their ease-of-use and have a reputation as expensive high-class smartphones. A wish for usability and a desire for prestige therefore make *two* reasons for buying one of them (cf. point 1 in the list above).

Both the wish for usability and the desire for prestige can be the *sole* reason a customer wants an Apple smartphone (cf. point 2). There may also be customers that want *both* an easily usable and prestigious phone but who do not want an Apple smartphone, e.g., because of company regulations (cf. point 3).

Last, but not least, Apple smartphones may also appeal to people that *neither* have a particular need for usability or prestige, since these are certainly not the only reasons for buying an Apple phone (cf. point 4).

Central to the Noisy-OR concept is the provisioning of an *inhibition probability* for each possible cause, representing the likelihood that the cause is unable to achieve its effect. In our approach, instead of specifying these probabilities directly, we calculate them by using the weights of the corresponding edges.

Definition 4.2.4 (Inhibition Probability Conversion Function) A strictly decreasing function $ipc : w \to]0, 1[$ is called *inhibition probability conversion function.*

In other words, greater weights w lead to smaller inhibition probabilities $ipc(w)$, which effectively "implements" definition 4.1.3. On the other hand, this general definition allows a great amount of freedom for implementers to adapt the function to their own interpretation of *weights* from their intermediate model.

Note that there is no conceptual restriction on the allowed values for a weight w, as long as the value can be processed using the conversion function. In practice, however, we found it useful to strictly limit the available values as we will show in example 4.2.6.

Definition 4.2.5 (Valuation of a Random Variable) Let V denote a random variable in a Bayesian network.

A set of tuples $val(V) = \{(P, p)\}$ with P a random variable and $p \in dom(P)$ is called a *valuation* of V, iff it contains exactly one tuple $(P, *)$ for every $P \in parents(V)$.

We now present an algorithm that calculates a probability distribution for a random variable given a concrete valuation of its parents in the Bayesian network. In order to extend the Noisy-OR approach to our multi-valued random variables, we calculate the inhibition probability individually for each possible value. We then normalize the probabilities to obtain a proper probability distribution for the variable. Recall that all random variables considered are assumed to have finite domains.

Algorithm 4.1 Calculation of a Probability Distribution for a Single Valuation $val(V)$

1: Let $ipc(w)$ be an inhibition probability conversion function.
2: Let V be the random variable for which the CPT is to be calculated.
3: Let $inhibition[]$ be an array with $|dom(V)|$ elements, representing the "inhibition probability" of each outcome.
4: Let $inhibition[v]$, with $v \in dom(V)$, be the array element corresponding to v.
5: **for all** $v \in dom(V)$ **do**
6: $\quad inhibition[v] = 0.5$ {Initialise with a "base" probability}
7: **end for**
8: **for all** $(P, p, V, v, w) \in parentedges(V)$ **do**
9: $\quad$ **if** $(P, p) \in val(V)$ **then**
10: $\quad\quad inhibition[v] = inhibition[v] * ipc(w)$
11: $\quad$ **end if**
12: **end for**
13: Let $cpt[]$ be the array holding the current column of the conditional probability table.
14: **for all** $v \in dom(V)$ **do**
15: $\quad cpt[v] = 1.0 - inhibition[v]$
16: **end for**
17: $\mathbf{P}_{val}(V) = normalize(cpt[])$

Algorithm 4.1 only calculates a part of the entire conditional probability table, namely a single "column" of the table if we consider a CPT representation analogous to example 4.2.2. We use $\mathbf{P}_{val}(V)$ as notation for this probability distribution. By iterating through all possible valuations of the parent variables of V, algorithm 4.1 may be used to fill the CPT for V column-by-column.

The initialisation of the inhibition probabilities array with starting values of 0.5 was chosen to be consistent for instances where no edges to the corresponding effect exist in the intermediate model. In those cases, the starting value would remain unchanged

throughout the execution of algorithm 4.1, making a "leak" probability of 0.5 a logical choice. In our approach, a probability of 0.5 models a "no information available" state of the system.

Example 4.2.6 (Inhibition Probability Conversion Function) In our project, the inhibition probability conversion was chosen as $ipc(w) = 0.3^w$ according to definition 4.2.4, based on the following considerations (note in particular that $w > 0$ holds for our use case):

To simplify our domain modelling, we only allowed "weak" and "strong" Influences, represented by numerical weights of 1 and 2, respectively. Also, we decided that a weak Influence should correspond to an inhibition probability of 30% and a strong Influence should correspond to 10% based on the notion that two weak Influences were supposed to be roughly equal in strength to one strong Influence.

Therefore, the uniform formula of 0.3^w was chosen to approximate these inhibition probabilities (leading to 9% for weight 2), while producing consistent results for all other fractional values that may occur during the generation of the intermediate model (cf. section 4.1.2).

Proofs

Before we can prove the correctness and completeness of the transformation of an intermediate model into a Bayesian network, we must represent the conditional probability tables as calculated by algorithm 4.1 in closed form.

We use the familiar notation of $e.w$ to denote the weight of en edge $e = (T, t, X, x, w)$.

Proposition 4.2.7 (Closed form of algorithm 4.1) Given a conditional probability table as constructed by algorithm 4.1, the conditional probability of the outcome $T = t$ given a valuation $val(T) = \{(S_1, s_1), \ldots, (S_n, s_n)\}$ can be formulated as

$$P(T = t | S_1 = s_1, \ldots, S_n = s_n) = \frac{1 - 0.5(\prod_{e \in E_t} ipc(e.w))}{\sum_{E_t \in E_T} (1 - 0.5(\prod_{e \in E_t} ipc(e.w)))} \tag{4.1}$$

with

$$\begin{aligned} E_t &= \{e \in E | e = (X, x, T, t, *) \wedge (X, x) \in val(T)\} \\ E_T &= \{E_t | t \in dom(T)\} \end{aligned}$$

Proposition 4.2.7 gives us a single probability $P(T = t|S_1 = s_1, \ldots, S_n = s_n)$. Hence, the entire conditional probability distribution $\mathbf{P}(T|parents(T))$ can be calculated by iterating through all combinations of the values in $dom(T)$ and all possible valuations of the parent variables of T

In other words, each application of proposition 4.2.7 fills exactly one "entry" of the CPT shown in example 4.2.2, whereas algorithm 4.1 produces an entire "column" of the CPT during each invocation. We will now show that both ways are equivalent.

Proof The proof will first detail both the construction of the *inhibition*[] array and the composition of the set E_t:

Lines 5–7 initialise all elements of *inhibition*[] with the value 0.5, representing a "base" inhibition probability which will be decreased whenever a corresponding edge has to be considered.

The for-loop in line 8 iterates over *all* edges ending in V in the intermediate model. Line 9 checks whether the current edge corresponds to the currently constructed valuation $val(V)$.

If this is the case, the current edge e is known to be part of E_t and in line 10, the corresponding element of *inhibition*[] is multiplied with $ipc(e.w)$ with $e.w$ being the weight of e.

Therefore, after completion of the loop in lines 8–12, we have processed the contents of all sets E_x for all $x \in dom(V)$. Also, we know that each element of *inhibition*[] contains a product consisting of the factor 0.5 and one factor of $ipc(e.w)$ for each element of the corresponding set E_x:

$$\forall x \in dom(V) : inhibition[x] = 0.5(\prod_{e \in E_x} ipc(e.w)) \tag{4.2}$$

Now, since the elements of *inhibition*[] contain inhibition probabilities that we cannot use directly in the specification of a Bayesian network, we use lines 14–16 to fill the *cpt*[] array with the probabilties of the respective complementary events:

$$\forall x \in dom(V) : cpt[x] = 1.0 - (0.5(\prod_{e \in E_x} ipc(e.w))) \tag{4.3}$$

With E_T as defined in proposition 4.2.7, the *sum* of all elements in $cpt[]$ can be written as:

$$\sum_{x \in dom(V)} cpt[x] = \sum_{E_t \in E_T} (1 - 0.5(\prod_{e \in E_t} ipc(e.w))) \tag{4.4}$$

Since the sum may be unequal to 1.0, we normalize the array in line 17 to finally obtain the conditional probability $P(T = t|S_1 = s_1, \ldots, S_n = s_n)$ by dividing the corresponding element of $cpt[]$ by the sum:

$$P(T = t|S_1 = s_1, \ldots, S_n = s_n) = \frac{1 - 0.5(\prod_{e \in E_t} ipc(e.w))}{\sum_{E_t \in E_T} (1 - 0.5(\prod_{e \in E_t} ipc(e.w)))} \tag{4.5}$$

Hence, after line 17, we have calculated one entry of the probability distribution $\mathbf{P}_{val}(T)$ for a given valuation *val*. $\mathbf{P}_{val}(T)$ may therefore be constructed by iterating through the elements of $dom(T)$.

q.e.d.

In propositions 4.2.8 and 4.2.9 we will now show that changes in the intermediate model, namely adding an edge or modifying the weight of an edge, indeed have the expected effects on the generated conditional probability tables.

Proposition 4.2.8 (Soundness of algorithm 4.1, part a) Let $(\mathbf{V}, \mathbf{E}_{with})$ and $(\mathbf{V}, \mathbf{E}_{without})$ denote two intermediate models that are identical except for one arbitrary additional edge, i.e., $\mathbf{E}_{with} = \mathbf{E}_{without} \cup \{(S, s, T, t, w)\}$.

Then, the following inequality holds for the Bayesian networks generated from them:

$P_{(\mathbf{V},\mathbf{E}_{with})}(T = t|S = s, \ldots) > P_{(\mathbf{V},\mathbf{E}_{without})}(T = t|S = s, \ldots)$

Proof According to proposition 4.2.7, it holds:

$$P(T = t|S = s, \ldots) = \frac{1 - 0.5(\prod_{e \in E_t} ipc(e.w))}{\sum_{E_t \in E_T} (1 - 0.5(\prod_{e \in E_t} ipc(e.w)))} \tag{4.6}$$

The numerator of the right-hand side of equation 4.6 is also contained as one of the summands in its denominator.

We single out that summand and substitute $x_t := 1 - 0.5(\prod_{e \in E_t} ipc(e.w))$:

$$P(T = t|S = s, \ldots) = \frac{x_t}{x_t + \sum_{E_{t'} \in E_T \setminus E_t} (1 - 0.5(\prod_{e \in E_{t'}} ipc(e.w)))} \tag{4.7}$$

To simplify, we further substitute $c := \sum_{E_{t'} \in E_T \setminus E_t} (1 - 0.5(\prod_{e \in E_{t'}} ipc(e.w)))$:

$$P(T = t|S = s, \ldots) = \frac{x_t}{x_t + c} = \frac{x_t + c - c}{x_t + c} = 1 - \frac{c}{x_t + c} \tag{4.8}$$

Note that the value of c is not dependent on whether we calculate $P_{(\mathbf{V}, \mathbf{E}_{with})}(T = t|S = s, \ldots)$ or $P_{(\mathbf{V}, \mathbf{E}_{without})}(T = t|S = s, \ldots)$.

Conversely, $x_t = 1 - 0.5(\prod_{e \in E_t} ipc(e.w))$ differs between $(\mathbf{V}, \mathbf{E}_{with})$ and $(\mathbf{V}, \mathbf{E}_{without})$ by one additional factor of $ipc(e.w)$. Therefore, it holds that $x_t^{with} > x_t^{without}$ (since $ipc(e.w) \in]0, 1[$), leading to

$$1 - \frac{c}{x_t^{with} + c} > 1 - \frac{c}{x_t^{without} + c} \tag{4.9}$$

q.e.d.

Proposition 4.2.9 (Soundness of algorithm 4.1, part b) Let $\mathbf{E}$ denote a set of edges in an intermediate model and $\delta > 0$.

Let $(\mathbf{V}, \mathbf{E}_{big})$ and $(\mathbf{V}, \mathbf{E}_{small})$ denote two intermediate models that differ only in the weight of a single edge, i.e., $\mathbf{E}_{big} = \mathbf{E} \cup \{(S, s, T, t, w + \delta)\}$ and $\mathbf{E}_{small} = \mathbf{E} \cup \{(S, s, T, t, w)\}$.

Then, the following inequality holds for the Bayesian networks generated from them: $P_{(\mathbf{V}, \mathbf{E}_{big})}(T = t|S = s, \ldots) > P_{(\mathbf{V}, \mathbf{E}_{small})}(T = t|S = s, \ldots)$

The proof of proposition 4.2.9 is largely analogous to proving proposition 4.2.8. It exploits the fact that one of the $ipc(e.w)$ factors within x_t^{big} has a greater $e.w$ parameter than its corresponding factor in x_t^{small}, leading again to $x_t^{big} > x_t^{small}$ (since $x_t = 1 - 0.5(\prod_{e \in E_t} ipc(e.w))$ and $ipc(e.w)$ is strictly decreasing).

We have now proven the *soundness* of algorithm 4.1 in propositions 4.2.8 and 4.2.9 in the sense that modifications of intermediate models are appropriately represented in the Bayesian networks generated from them.

Additionally, we will prove its *completeness*, i.e., that if probability distributions in the generated Bayesian network deviate from the uniform distribution assumed in the default case, the reason of this *must* have been represented in the intermediate model.

Proposition 4.2.10 (Completeness of algorithm 4.1) Let $\mathbf{1} = (\mathbf{V}_1, \mathbf{E}_1)$ and $\mathbf{2} = (\mathbf{V}_2, \mathbf{E}_2)$ be two intermediate models. Assume without loss of generality that $\mathbf{V}_1 = \mathbf{V}_2 = \mathbf{V}$.

If, in the Bayesian networks generated from the models, $P^{\mathbf{1}}(T = t|S = s, \ldots) > P^{\mathbf{2}}(T = t|S = s, \ldots)$, then the following holds:

$$
\begin{array}{ll}
(i) & \exists e = (S, s, T, t, w) \in \mathbf{E}_1 : e \notin \mathbf{E}_2 \quad \vee \\
(ii) & \exists e_1 = (S, s, T, t, w_1) \in \mathbf{E}_1, \exists e_2 = (S, s, T, t, w_2) \in \mathbf{E}_2 : w_1 > w_2 \quad \vee \\
(iii) & \exists e = (S, s, T, t', w) \in E_2 : t' \neq t, e \notin \mathbf{E}_1 \quad \vee \\
(iv) & \exists e_2 = (S, s, T, t', w_2) \in E_2 : \exists e_1 = (S, s, T, t', w_1) \in \mathbf{E}_1 : t' \neq t, w_2 > w_1
\end{array}
$$

In other words, to account for the difference, either there exists an additional edge to t in $\mathbf{E}_1$ (i), one of the common edges to t of $\mathbf{E}_1$ and $\mathbf{E}_2$ have a different weight (ii), there exists an additional edge in $\mathbf{E}_2$ towards $t' \in dom(T) \setminus \{t\}$ (iii), or $\mathbf{E}_1$ and $\mathbf{E}_2$ have a common edge from s to t' that has a different weight (iv).

Proof Let $P^{\mathbf{1}}(T = t|S = s, \ldots) > P^{\mathbf{2}}(T = t|S = s, \ldots)$ with

$$
\begin{aligned}
P^{\mathbf{1}}(T = t|S = s, \ldots) &= \frac{1 - 0.5(\prod_{e \in E_t^{\mathbf{1}}} ipc(e.w))}{\sum_{E_v^{\mathbf{1}} \in E_T^{\mathbf{1}}} (1 - 0.5(\prod_{e \in E_v^{\mathbf{1}}} ipc(e.w)))} \\
P^{\mathbf{2}}(T = t|S = s, \ldots) &= \frac{1 - 0.5(\prod_{e \in E_t^{\mathbf{2}}} ipc(e.w))}{\sum_{E_v^{\mathbf{2}} \in E_T^{\mathbf{2}}} (1 - 0.5(\prod_{e \in E_v^{\mathbf{2}}} ipc(e.w)))}
\end{aligned}
$$

Both conditional probabilities are fractions with numerator and enumerator each > 0. Hence, for the inequality to hold, it must be the case that either **a)** the *numerator* of $P^{\mathbf{1}}(T = t|S = s, \ldots)$ is greater than the numerator of $P^{\mathbf{2}}(T = t|S = s, \ldots)$ or **b)** its denominator is smaller – or both.

a) Numerator

Assume that the numerator of $P^{\mathbf{1}}(T = t|S = s, \ldots)$ is greater and simplify:

$$\begin{aligned} 1 - 0.5(\prod_{e \in E_t^{\mathbf{1}}} ipc(e.w)) &> 1 - 0.5(\prod_{e \in E_t^{\mathbf{2}}} ipc(e.w)) \quad \Leftrightarrow \\ \prod_{e \in E_t^{\mathbf{1}}} ipc(e.w) &< \prod_{e \in E_t^{\mathbf{2}}} ipc(e.w) \end{aligned}$$

which may be the case if the left-hand side consists of more factors than the right-hand side (since $ipc(x) \in]0, 1[$) or if at least one factor on the left-hand side has a greater $e.w$ parameter than the matching factor on the right-hand side (due to the monotonicity of $ipc(x)$).

Since the construction method ties each factor and its exponent to exactly one edge and its weight, this leads to (i) or (ii) in proposition 4.2.10.

b) Denominator

Assume that the denominator of $P_{\mathbf{1}}(T = t|S = s, \ldots)$ is smaller and simplify:

$$\begin{aligned} \sum_{E_v^{\mathbf{1}} \in E_T^{\mathbf{1}}} (1 - 0.5(\prod_{e \in E_v^{\mathbf{1}}} ipc(e.w))) &< \sum_{E_v^{\mathbf{2}} \in E_T^{\mathbf{2}}} (1 - 0.5(\prod_{e \in E_v^{\mathbf{2}}} ipc(e.w))) \quad \Leftrightarrow \\ \sum_{E_v^{\mathbf{1}} \in E_T^{\mathbf{1}}} \prod_{e \in E_v^{\mathbf{1}}} ipc(e.w) &> \sum_{E_v^{\mathbf{2}} \in E_T^{\mathbf{2}}} \prod_{e \in E_v^{\mathbf{2}}} ipc(e.w) \end{aligned}$$

Since $ipc(x) \in]0, 1[$, all factors are > 0 and therefore all summands are > 0. Hence, the inequality holds only if

$$\exists v \in dom(T) \quad : \quad \prod_{e \in E_v^{\mathbf{1}}} ipc(e.w) > \prod_{e \in E_v^{\mathbf{2}}} ipc(e.w)$$

which, analogously to the line of reasoning in a), can only be the case if the product $\prod_{e \in E_v^{\mathbf{1}}} ipc(e.w)$ has *fewer* factors than $\prod_{e \in E_v^{\mathbf{2}}} ipc(e.w)$ or that one or more factors have *smaller* $e.w$ parameters (due to the monotonicity of $ipc(x)$).

Again, each factor has exactly one corresponding edge in the intermediate model. Hence, the existence of additional edges towards v or variations in their weights can be deduced. However, two cases regarding v must be distinguished for such an edge:

1. $v \neq t$:
   In this case, the corresponding edge is not connected to t but to a different element of $dom(T)$, leading to (iii) or (iv) in proposition 4.2.10, respectively.

2. $v = t$:
   Here, the corresponding edge would be connected to t. However, according to propositions 4.2.8 and 4.2.9, the existence of such an edge in $\mathbf{E}_2$ would lead to $P^{\mathbf{1}}(T = t|S = s, \ldots) < P^{\mathbf{2}}(T = t|S = s, \ldots)$, violating the initial assumption of proposition 4.2.10. Therefore, this case is impossible.

q.e.d.

To summarize, these proofs show both the soundness and completeness of the transformation of an intermediate model into a Bayesian network according to algorithm 4.1. Hence, it can be said that the generated Bayesian networks adequately capture the semantics of a domain model according to definition 3.3.4.

4.3 Conclusion

In this chapter, we have have shown how the Bayesian network that forms the core of the inference engine of our approach is generated from a domain model. To this end, the model is first transformed into a more detailed and formalized intermediate representation. From there, the Bayes network is generated in such a way that its graph structure and calculated conditional probability tables reflect the semantics of Influences as established in definition 4.1.3. The construction of the conditional probability tables is based on the "Noisy-OR" concept which was adapted to suit the requirements of the use case adequately.

While the transformation of a domain model into its intermediate representation is relatively straight forward, we have explicitly proven the soundness and the correctness of the algorithm that generates the Bayesian network. Adding the intermediate layer creates a certain abstraction between the domain model and the Bayesian implementation that allows to keep adaptations of the domain model semantics local and independent of the Bayesian network, as long as the intermediate representation can remain constant (see also figure 1.2).

5 Dialogue Management

5.1 Question Based Preference Elicitation

Our general approach is to elicit the customer's preferences by having the recommender system pose questions that the customer is supposed to answer. While it is not our goal to incorporate natural language processing in the narrower sense, the dialogue *structure* must be flexible enough to support a natural dialogue flow, which is crucial to our use case of supporting a salesperson in his/her recommendation dialogue directly at the point-of-sale.

Therefore, a rigid dialogue structure would not be acceptable for both salespersons and customers, leading us to formulate the following requirements for the dialogue management:

- *Answering a particular question must not be compulsory.*
  It must be possible to skip questions and the recommender system must be able to seamlessly continue its dialogue. Likewise, the system must not rely on a "completed" dialogue before providing recommendations. Ideally, recommendations should be provided for all combinations of answered and unanswered questions.
- *A customer must be allowed to change the topic on his/her own initiative.*
  The dialogue manager must be able to respond to cases where the customer does actually answer a different question than the current one.
- *A customer must be allowed to modify his/her answer to a past question.*
  Such a belief revision must be possible without interference to other parts of the dialogue, e.g., without losing answers to later questions (as would commonly be the case in a "wizard" style dialogue with a strict forwards and backwards navigation paradigm), and the changed knowledge must be integrated into the recommendations transparently.

To fulfill these requirements, the dialogue management relies to a certain extent on an inference engine whose features (e.g., to support the partial knowledge that must be assumed if users are allowed to skip questions) will be discussed in chapter 6.

The goal of this chapter is to describe how the dialogue is built to take advantage of those features.

We use a set of statecharts (cf. section 2.3) which are in part domain-independent and in part domain-dependent to both visualize the dialogue structure and to provide well-defined execution semantics for the system's behaviour for implementors (see section 5.3 for notes on our implementations). This approach has been published in [RBF07].

5.1.1 General Interaction Concept

Fig. 5.1 shows a coarse overview of the general dialogue cycle that we will refine during the course of this chapter. Esssentially, the recommendation process is divided into two cyclic sub-processes that run in parallel:

- The *recommendation loop* continually alternates between presenting a recommendation to the customer and producing a new, updated recommendation for display.
- The *dialogue loop* similarly switches between choosing the optimal next questions and presenting them to the customer, waiting for an answer.

As shown in the figure, the loops are synchronised by the customer's answers – both have to react to the newly acquired knowledge which is signalled by the answer event and, on the other hand, cannot do meaningful work in the background without new knowledge.

Modelling both loops independently of each other allows a greater flexibility for implementors. A suitable adaptation of the statechart might, for example, remove the strict requirement that every answer produces a new recommendation and opt to update the recommendations only if the inference engine detects significant changes in its estimations.

Also, the modelling hints at a parallelism that may be leveraged by implementors if we think of, e.g., AJAX-based web applications that can query a server using asynchronous parallel requests.

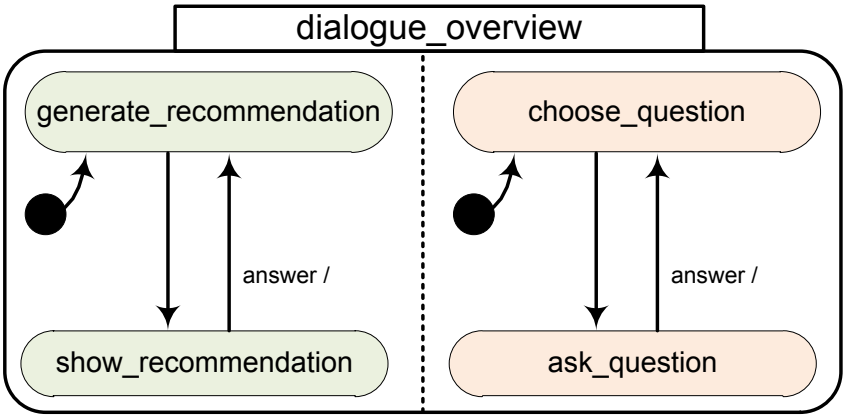

Figure 5.1: General dialogue overview

In order to reach an appropriately fine-grained representation of the dialogue, we will begin by refining the representation of the recommender cycle (i.e., the left-hand side of Fig. 5.1). Although it is the goal of our system to provide useful recommendations as early as possible, the acquired knowledge at a given point in the dialogue may be insufficient to do so – most notably this would likely be the case before any questions have been answered.

Therefore, Fig. 5.2 extends the correponding statechart by allowing the system to choose an alternative default recommendation to be displayed. A suitable default could be built, e.g., from the current top-selling products, based on advertising, or, as will be detailed in section 6.3, the inference engine may be used despite the low knowledge, under some circumstances. As before, the answer event triggers a re-evaluation of the situation based on newly elicited knowledge.

Secondly, the dialogue cycle needs to be structured in more detail. To this end, the questions are organised into a number of layers, as detailed below.

5.1.2 Dialogue Structure

Before we describe the dialogue structure itself, we should establish the source of the questions that will be used in the dialogue. Basically, the question pool is formed by creating questions for the ReasoningElements in the domain model at hand. In fact, we already hinted at this in example 3.1.1, by giving a question wording for the Needs defined there.

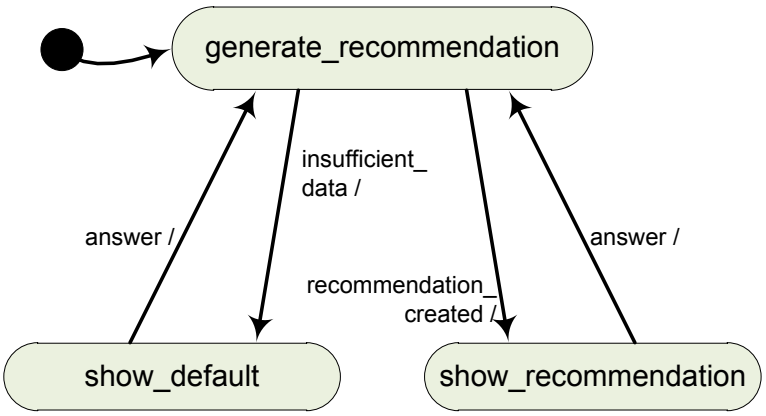

Figure 5.2: Refined recommendation cycle

The exact way of creating questions depends on the subtype of the concrete ReasoningElement:

- For Traits contained in a TraitGroup, one single question for the complete TraitGroup is generated, with corresponding answer options for all Traits in the TraitGroup, implementing the "single choice" semantic established in section 3.1.
- For a stand-alone Trait, a question with the answer options "positive" and "negative" is generated.
- For a Need, a question with answer options according to parameter "detail" introduced in section 4.1.2 is generated.
- For AttributeValues, one question for the corresponding Attribute is generated, with answer options according to the all AttibuteValues.

More formally, a question q is defined via its associated set R_q and A_q as follows:

Definition 5.1.1 (Questions) Let q denote a question. Then R_q denotes the set of ReasoningElements and A_q denotes the set of answer options that correspond to q.

We do not formally define question wordings since their exact composition depends on the concrete application architecture (and, particularly, user interface design), but remember that questions will commonly have natural texts and/or images associated with them that correspond to the question itself and its answer options.

Now, these dialogue questions are organised as a number of layers that are supposed to

correspond to the customer's perceived difficulty of answering them. Since we do not model a question difficulty explicitly, the type of the ReasoningElement corresponding to the question is used based on the following considerations:

Questions for Traits are supposed to be the easiest to answer since they only elicit objectively determinable information. Needs form the second layer, requiring somewhat more effort for being answered. Finally, questions about Attributes are considered the most difficult. Since we cannot generally assume that a customer is at all able to express his/her technical preferences, these questions should preferably be completely avoided. On the other hand, some customers may have particular technical wishes or the dialogue may have provided only inconclusive information about a customer. Hence, the dialogue manager should be able to transition to the most technical layer if required or requested by the customer.

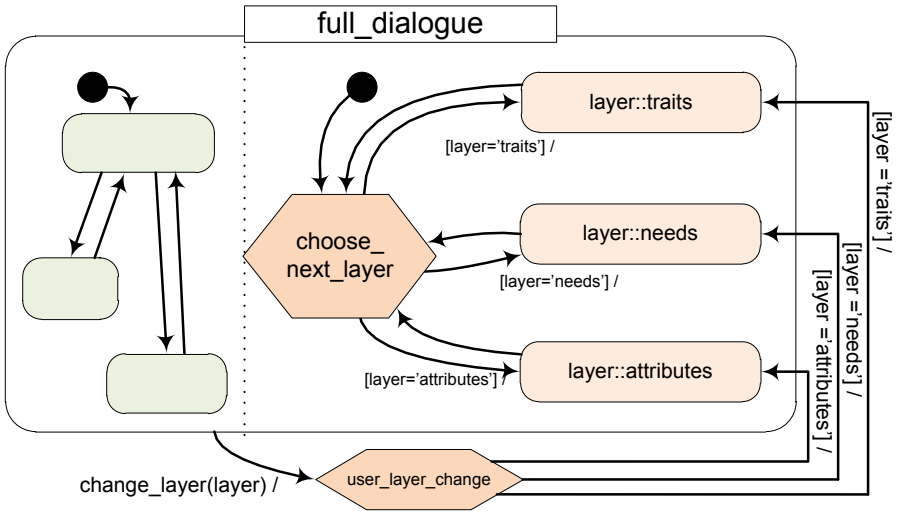

Figure 5.3: Dialogue layers

Fig. 5.4 details the extensions to the overview statechart necessary to accomodate the different layers in the dialogue. The decision state choose_next_layer is used by the dialogue manager to select the next dialogue level based on information from the inference engine (cf. section 6.1), whereas a customer is able to switch to the layer of his/her liking by triggering the change_layer event – something which can be modelled very conveniently in the statechart ECA-syntax.

The mechanism of user- and system-initiated changes of difficulty can be generalized to cover multiple dialogue *topics* as well, e.g., if the recommender system is built to

combine several different domains into one large domain model as hinted to in section 3.4. Fig. 5.4 illustrates this more general design. In such a case, a topic itself may be composed of several layers or sub-topics, as required for the domain(s) at hand.

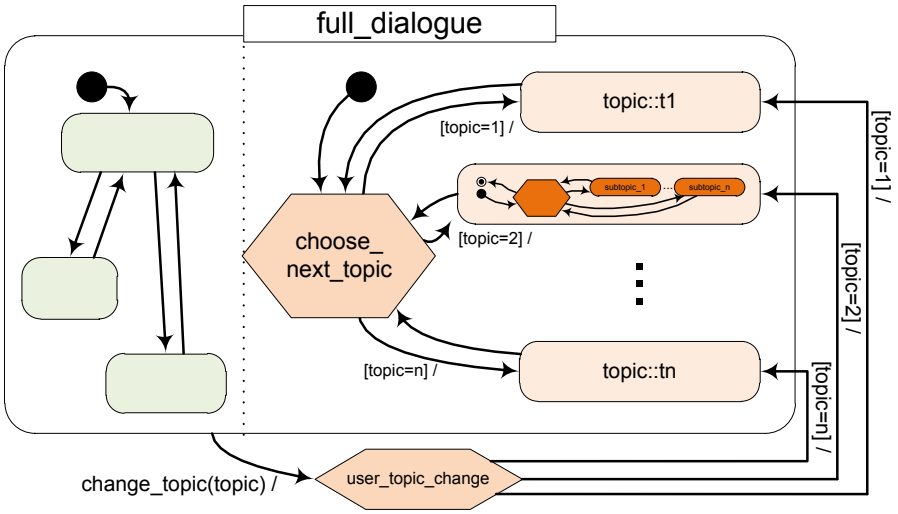

Figure 5.4: Dialogue with multiple topics

5.1.3 Layer / Topic Structure

Up to this point, all statecharts were largely domain-independent, without looking at the particular pool of questions that is available for the domain at hand. As a reminder, we have established that every ReasoningElement of a domain model is the source for a question and that these questions are organised into layers/topics according to their subclass.

Now, we show the "deeper" structure of any given layer, realised as hierarchical states in our syntax. In other words, even while the system may be in a deeply nested state, all transitions at the upper levels can be triggered as well. This becomes relevant, e.g., for the change_layer event described in the preceding section which allows switching to a different layer no matter what question is currently asked. The so-defined behaviour can be compared to the "exception" mechanisms known in many common programming languages.

Within a layer, the structure is quite similar to the topmost group of states for the different layers. A layer consists of one decision state that allows the dialogue manager

to decide which question to ask next and one state for every question in the layer as shown in fig. 5.5. Again, a special event caters for user-initiated changes of the current question, enabling a fully local reaction to this situation without further measures.

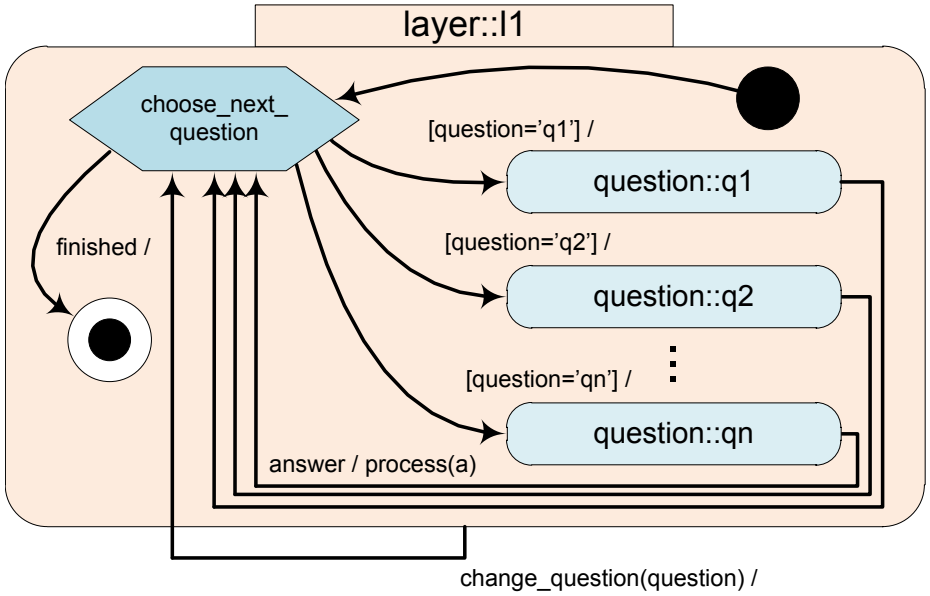

Figure 5.5: Sub-statechart for a single layer

Note that the dialogue manager has the option to transition into an explicit end state within a layer. If this occurs, the control flow of the statecharts returns to the hierarchy level above, which, according to fig. 5.3, would then initiate the selection of a new layer to continue the dialogue.

Often, a user interface requires that not only a single question be asked at one point in time but rather that the user has the option to answer several questions *simultaneously*. However, statecharts do not provide a native means of specifying that a system is supposed to be in "n-out-of-m" states at the same time. Hence, in order to support this behaviour, we have to extend the generated statecharts (as illustrated by fig. 5.6 in comparison to fig. 5.5):

- An AND-state (cf. section 2.3.2) with a number of sub-states equal to the maximum number of simultaneously asked questions is introduced.
- Each of the sub-states of the AND-state contains one state per question in the dialogue.

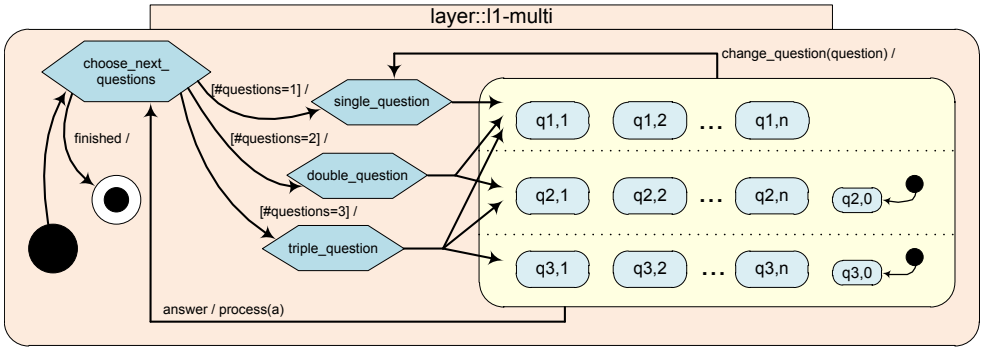

Figure 5.6: Single layer with 1-3 simultaneous questions (some hyperedges and labels omitted for readability)

- All sub-states except the first contain another state representing "no question" connected to a starting state. This is done in order to support the possibility that less than the maximum allowed number of questions is displayed in a given dialogue step. E.g., if a dialogue supports up to three simultaneous questions but a given dialogue step only contains two questions, the third sub-state transitions into the "no question" state.
- Additional decision-states are introduced to differentiate between the number of questions in a given dialogue step. From these, the appropriate hyperedges are generated to allow every possible combination of questions for a dialogue step by transitioning into the corresponding states within the large AND-state.

Despite being somewhat cumbersome because of a potentially big number of states and hyperedges, the extension is straight-forward and in particular keeps the simple answer-based synchronization of the dialogue steps.

Example 5.1.2 (Complete statechart for the running example) Our running example (3.3.7) is small enough so that it can be presented as a statechart in its entirety (although we already have to omit some labels). According to section 5.1.2, the domain-dependent part or the statechart will be composed of three layers:

1	Traits	ageclasses
2	Needs	multimedia, music, office, internet
3	Attributes	mp3, memory, umts

Fig. 5.7 shows the complete generated statechart. Accord to the figure, when the recommendation process is started, the dialogue manager will transition into the

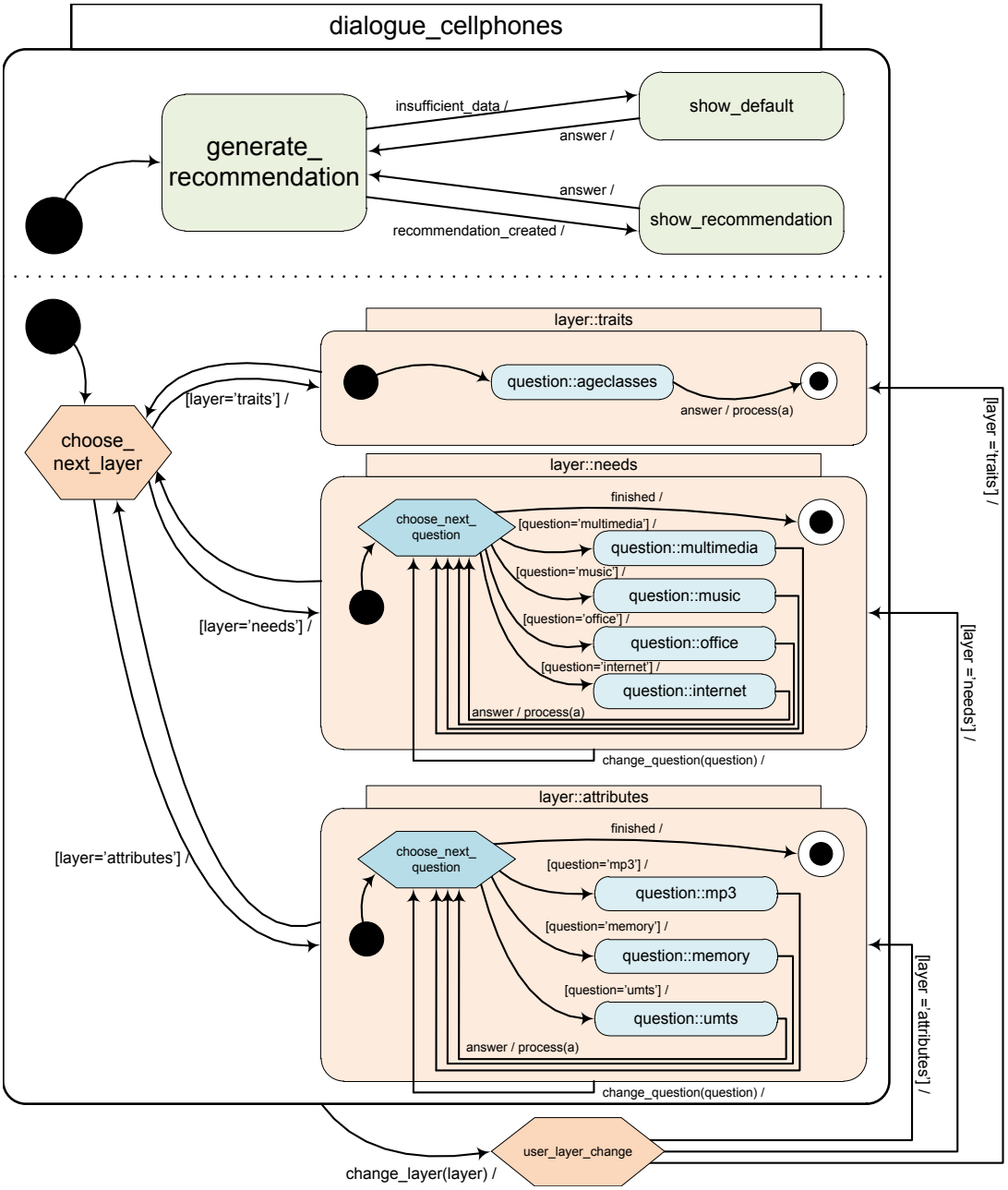

Figure 5.7: Statechart for the running example

show_default state for the recommendations and into the question_ageclasses state for the dialogue. The dialogue manager will use the layer::traits layer as it is supposedly the easiest for the customer to answer. And within that layer, the only available question is question_ageclasses (for this reason, layer::traits also uses a simplified structure that leaves out the normally necessary decision state).

Once the customer gives an answer to the first question, the current recommendation will be created and displayed in the show_recommendation state. In parallel, layer::traits will be completed, the control flow will reach choose_next_layer and continue into layer::needs where the next question that will be chosen by the dialogue manager will depend on the concrete answer previously given by the customer.

5.2 Question Relevance

One of the principal tasks of the dialogue manager is to determine the next step of the dialogue by evaluating the available questions regarding their usefulness for the recommendation process. When reasoning about the optimal next question in our framework, the system has two conflicting goals:

- On the one hand, it is important for a recommender system that the customer trusts its recommendations. This is particularly important for our use case since the interactions between customer and recommender are very limited (i.e., there is no long-lasting customer relationship as would frequently be assumed for collaborative filtering approaches). Therefore, practically the only chance for the recommender system to demonstrate understanding and expertise to the customer is during the sales dialogue.
  We can accomplish this by using the Bayes network to predict the customer's answers to future questions and to use these predictions to customise the questions, e.g., by choosing a more suggestive wording or by highlighting the customer's most probable answer. Obviously, these measusures can only be successfully employed if the predictions are sufficiently *certain*, because incorrect predictions would be counterproductive for the goal of presenting a recommender system that understands the customer.

- On the other hand, sales dialogues are supposed to be as short as possible in order to increase conversion rates. To this end, the recommender system must provide meaningful recommendations quickly (i.e., after few dialogue steps), which makes it necessary to prioritise questions that promise a large gain in knowl-

edge. On average, the greatest knowledge gain will be achieved for questions whose predictions are particularly *uncertain* since answering such questions is guaranteed to increase the system's knowledge. On contrast, answering questions with highly confident predictions is likely to only confirm them, without acquiring new knowledge.

Based on these considerations, a question's *relevance* for the dialogue process should be connected to the confidence of the answer prediction for the question. In our metamodel, the ReasoningElements have an associated BeliefState, which, according to definition 3.3.2 contains the "degree of belief" of the two possible states of a ReasoningElement, β^+ and β^-.

The precise way to calculate these values will be detailed in section 6.3. At this point, however, it is just necessary to define how the relevance of a question relates to β^+ and β^-.

The core question, i.e., whether to favour questions with highly confident or deliberately un-confident predictions, remains. To get more insights into this, we conducted a small informal experiment during the University of Passau Open Day in 2008 where a prototypical recommender system was deployed to recommend courses of study to visiting interested students.

There, questions were asked in a way that preferred confident predictions, i.e., to test how well the predictions were accepted. To this end, the system visualised its predictions by pre-selecting the most likely answer option. The evaluation was carried out informally by observing the users and casually interviewing them after the recommendation process was complete.

Our experiment yielded interesting results:

- Predictions were frequently accepted (i.e., the pre-selected answer option was rarely changed).
- The system was generally regarded as being "competent" regarding the predictions. The so-earned trust carried over to the generated recommendations.
- The early questions frequently did not have a great influence on the generated recommendations. On the other hand, answering the final questions commonly changed the recommendation significantly.

Clearly, the last property is undesired – even more so if we consider that the recommendations were only perceived as "correct" at the end of the dialogue. Fortunately,

in the experimental setup, the users were mostly eager to "play" with the system and follow the dialogue until the end. This cannot be assumed for an actual sales use-case, however, and there exists a significant concern that the system would not be able to produce good recommendations in time before prospective customers grow inpatient and leave.

As a consequence, we decided that the advantages of asking confidently predicted questions was offset by the poor recommendation quality early in the dialogue. Question relevance is therefore defined as follows, favoring fast knowledge-gains:

Definition 5.2.1 (Relevance of a Question) Let q denote a question and R_q denote the ReasoningElements corresponding to q as in definition 5.1.1.

Then, the *relevance* $rel(q)$ of q is defined using the average difference between the two degrees of belief β_r^+ and β_r^- of each ReasoningElement $r \in R_q$ to model the confidence:

$$rel(q) = 1 - \frac{\sum_{r \in R_q} |\beta_r^+ - \beta_r^-|}{|R_q|}$$

A higher value of $rel(q)$ corresponds to a lower confidence in the predictions (i.e., the degrees of belief are very similar) and means that the question is more relevant to the current state of the recommendation dialogue (i.e., should be asked earlier).

5.3 Implementation

Statecharts as described in section 5.1 have precise execution semantics. If created at a suitable degree of detail, executable code may be obtained from statecharts, e.g., by techniques of the Model Driven Architecture ("MDA" – cf. [Fra03]), or by directly executing a statechart through a generic interpreter.

However, instead of generating the very detailed statecharts necessary for such an approach, we decided to implement the domain-independent part of the statecharts as a dedicated web application that shows behaviour as implied by the statecharts. The domain-dependent parts (i.e., the question lists and their structuring into layers / topics) are generated and then used as parameters in the web application. Figure 5.8 illustrates this architecture by showing a corresponding UML diagram.

The interaction model of a web application fits the statechart design well: In our statecharts, the only externally caused events are answer, change_question, and change_layer. In the view of the web-based implementation, these events are seen in the form of "clicks" (or, more precisely, HTTP requests) caused by the user, that move the control

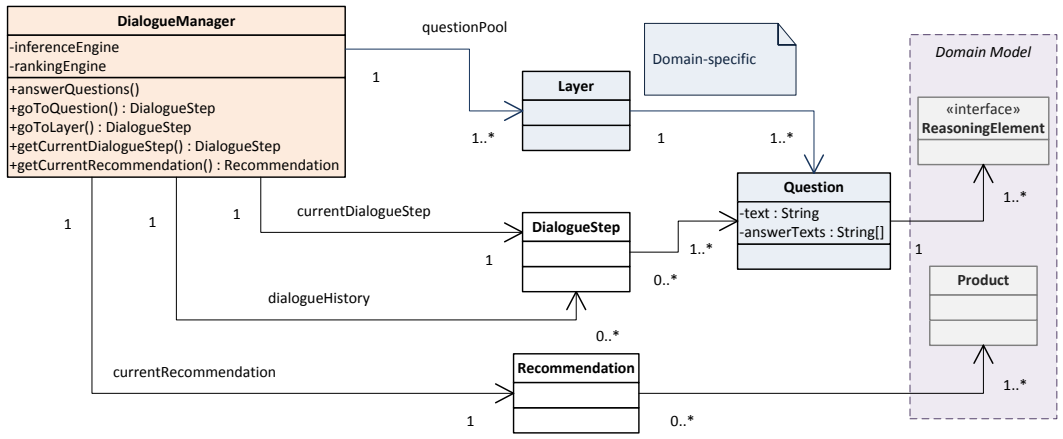

Figure 5.8: UML diagram for the dialogue management component

flow from states that are on the client side to states on the server side, as conceptualised in figure 5.9. The server application then decides about the next dialogue step and delivers a HTML page to the client, which is equivalent to a transition into the states on the client side and waiting for the next event.

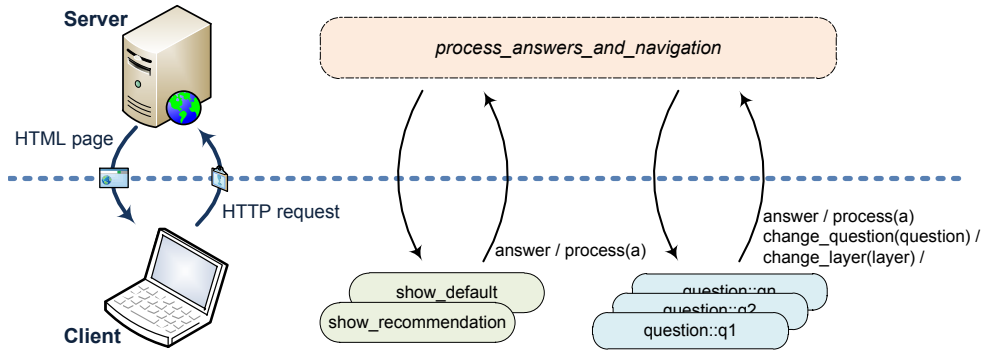

Figure 5.9: Statecharts implemented as a web application with HTTP requests for the external events

Finally, it is noteworthy that the implementation is not limited to offering a web application. The DialogueManager and its associated classes shown above form the model of a system architecture based on the Model-view-controller ("MVC") software architecture (cf., e.g., [BD04, chapter 6]). In our current implementation, controller and view are implemented as Java Server Pages ("JSPs") and Java servlets, respectively, but the model is fully independent of this technology.

In fact, in the cooperation with our industry partner, the same architecture was employed to build a Service-Oriented Architecture ("SOA" – cf. [Erl06]) where a "rich" client application accesses the dialogue management and recommendation functionalities of a web service whose backend is designed very similar to figure 5.8.

5.4 Conclusion

In this chapter, we showed a dialogue modelling based on statecharts and a web-based implementation of the implied system behaviour that fulfill the requirements posed towards a dialogue management component of a conversational recommender system. The main contribution of our method is its flexibility that allows for using the system even in highly dynamic environments such as during a natural sales dialogue. Assisting salespersons directly at the point-of-sale is explicitly part of the design goals of our system.

In particular, neither the modelling nor the implementation rely on fixed paths through the dialogue. Therefore, no question must be seen as compulsory for a continuation of the recommendation process. By de-coupling the dialogue management and the generation of recommendations, the system is able to react to any new answers with an updated recommendation.

As such, the dialogue does not have to be completed before recommendations can be displayed. However, the modelling does allow for the option of displaying a default recommendation in the case of insufficient knowledge. We will show in section 6.3 why this is rarely necessary in our approach, if at all.

Also, the model and implementation support user-initiated changes of topic and allow to change the question within the current topic. Nothing precludes a customer to use this mechanism to return a a previously answered question and to re-answer it differently.

The synchronisation of the recommendation component to the dialogue progress based on the answer event ensures that the so-changed knowledge is taken into account automatically, without having to, e.g., backtrack the dialogue history and invalidating other answers (in part, this is a feature of the inference engine based on Bayesian networks that we describe in chapter 6 below).

Related Work

In its flexibility and ability to support user-initiated actions, our dialogue management is notably different from other conversational recommender system approaches, such as [AFF+02, AFF+03, FFJZ06]. Ardissono and Felfernig model the dialogue by specifying which question succeeds which other questions, forming a graph-like structure with all paths that the dialogue is allowed to take. The approach allows a very fine-grained, explicit modelling of the dialogue (up to the point of specifying additional user interface elements such as info boxes). On the other hand, the requirement for explicit specification naturally limits the possible freedom.

The systems based on this approach that we have seen deployed in practice[1] seem to only support a wizard-style "forward" / "backward" navigation through the dialogue. It is unclear how belief revision, skipping questions, and recommendations despite an incomplete dialogue can be handled comfortably. Furthermore, to suggest personalised default answers to questions, our approach does not need to rely on a set of business rules as appears to be the case in [AFF+02].

Statecharts were also proposed in [Köl99] as a means of modelling general conversational dialogues. However, her intention is to provide dialogue designers with an intuitive tool for modelling the dialogue, whereas we focus on generating the dialogue automatically, given a domain model of a particular area of application (namely, product recommendation) and use statecharts as a means of visualization and basis of further processing (e.g., code generation).

In [SB01], dialogues are formally modelled as state machines. In their model, a "situation" (state) contains information about the preferences of the user, the dialogue history and the current recommendation, resulting in a large space of possible situations. Depending on user input, different transition functions between these states are executed (called "interactions"). Our approach, in contrast, does not need to include preferences, recommendations etc. into the definition of the states themselves. Instead, this information is accessible via local variables of the statecharts at those points where it is needed, considerably reducing the complexity of the model.

Also, with the approach of [SB01], a dialogue developer has to define the "strategy" of the dialogue either by modelling a directed graph that combines the possible interactions with the possible situations (resembling, in a way, a statechart) or by providing a set of ECA rules. This approach is static and due to the presumable complexity of the model not applicable to the rapidly changing domains that our approach targets.

[1] Visit `http://www.premium-cigars.ch/index.php?action=mortimer` for an impression (in german).

It allows for a very flexible crafting of the dialogue, but, on the other hand, presents a correspondingly big challenge to the domain expert that is charged with the task of specifying the strategy. Alternatively, they propose dynamic techniques based on CBR to determine the next relevant question to be asked, e.g., by measuring information gain.

Another approach is presented in the work by the research group around Pearl Pu at EPFL [HP10, HP09]. They use standard personality quizzes to infer the properties of desired products. A central point of their approach is that no recommendation is possible if the dialogue is incomplete. Also, it remains doubtful for us whether a standardized psycholgical questionnaire is able to provide customer's preferences for general market domains.

The flexibility of our approach becomes particularly apparent if we imagine the system being used in a store to support a sales dialogue between natural persons, which is explicitly part of our use case. Whereas a certain rigidity in the dialogue flow may be acceptable when interacting with a computer program (since users simply do not expect anything better), it would be plainly inadequate in that context.

6 Inference Engine

6.1 Inference Engine Requirements

The inference engine is a core component of our recommender system, encapsulating the central "intelligence" functions for both managing the dialogue and recommending products. In summary, it must be able to provide the following functionalities:

- *Add new knowledge.*
  In our approach, this is accomplished by answering questions.
- *Revise previously acquired knowledge.*
  Questions may be re-answered by the customer.
- *Predict unanswered question responses.*
  This may be used to improve dialogue behaviour.
- *Evaluate the relevance of questions.*
  This is used for the system-led dialogue path.
- *Predict the usefulness of technical properties.*
  The main goal of our approach is to derive appropriate knowledge about the technical properties the customer desires. This encompasses prioritizing properties within single technical attributs as well as weighing attributes against each other.
- *Explain system behaviour.*
  This is used to provide feedback to the customer about why a certain product was recommended.

The following sections will show how a Bayesian network as described in section 4.2 can be used to implement the interface that is posed by the requirements. Figure 6.1 shows a UML representation of the implementation, containing the InferenceEngine itself and the BayesNetwork with their methods. Note in particular the simplicity of the required functions of the Bayesian network. For our concrete implementation

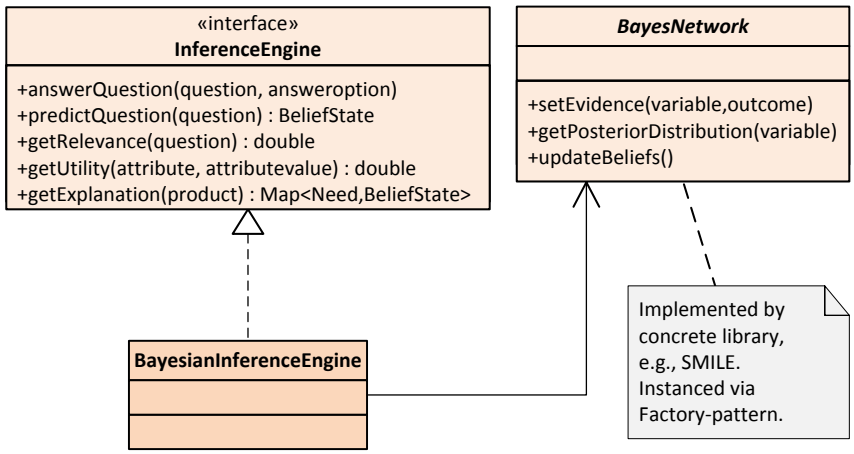

Figure 6.1: UML diagram of the inference engine

we use the SMILE library introduced in section 2.2.4 and its graphical user interface GeNIe.

6.2 Answers

As mentioned above, our recommender system acquires knowledge by processing the answers asked in the dialogue. There can be further sources to gain information from, such as querying data from a CRM system if applicable, but ultimately, these can also be broken down to answering the modelled questions.

In general, given an answered question q according to definition 5.1.1, its answer a is one of the available answer options, i.e. $a \in A_q$ and it is used in the form of *evidence* for the corresponding random variables in the Bayesian network. The exact way to treat an answer is, once again, depending on the ReasoningElement(s) corresponding to the question (cf. section 5.1.1):

- For questions referring to Traits contained in a TraitGroup, a denotes a particular Trait t_a and the following evidence is set in the Bayes net:
  - For r_{t_a}, i.e., the random variable corresponding to t_a, its outcome pos_1 is set as evidence.
  - For each other random variable relating to the TraitGroup, its outcome neg_1 is set as evidence.

- For a stand-alone Trait, a is either "positive" or "negative". Correspondingly, evidence is set for either the outcome pos_1 or neg_1, respectively.
- For a question referring to a Need, there ist a 1:1 mapping between answer options and outcomes of the corresponding random variable, and evidence is set accordingly.
- For questions relating to an Attribute, a denotes a particular AttributeValue and the following evidence is set in the Bayes net:
  - The random variable corresponding to the AttributeValue, its outcome pos_1 is set as evidence.
  - For the random variables relating to the other AttributeValues of the Attribute, their outcomes neg_1 are set as evidence.

After setting the appropriate evidence, the Bayesian network then has to update its calculations for the posterior probability distributions for all random variables that are still not known by evidence. Section 9.2 will provide performance measurements. Therefore, it suffices to state that, given the size of our Bayesian networks, we commonly have to rely on sampling algorithms (cf. section 2.2.4.2) to obtain results within time constraints appropriate for interactive operation.

Example 6.2.1 (Bayes network with evidence) Figure 6.2 shows the Bayesian network corresponding to the running example 3.3.7 with updated posterior probabilities for an answer of "agree strongly" (i.e., pos_2) for the question corresponding to the Need "office".

A Bayesian network has two significant properties that are particularly helpful for our use case: It does not place particular demands on the ordering in which evidence is inserted, nor does it rely on "immutability" of evidence once set, although some algorithms may profit from such a property by only having to do incremental re-calculations. These qualities are very accommodating to our requirements for conducting a flexible recommendation dialogue.

Based on these considerations, the mechanism for belief revision also becomes apparent: If the customer changes an answer, the evidence for the corresponding random variables is simply re-set to reflect the new answer. The new knowledge will then be used transparently. Again, Bayesian networks support this central requirement of our use case in a natural way.

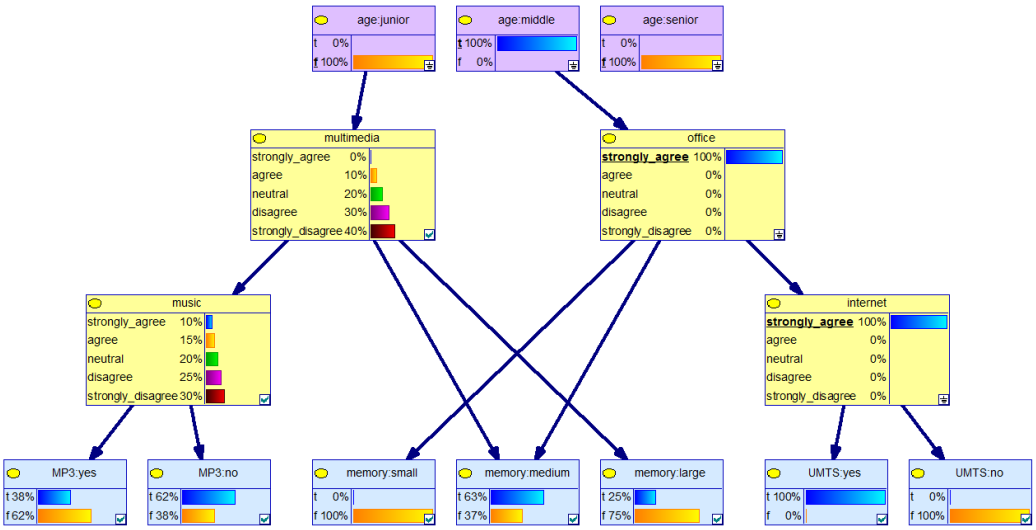

Figure 6.2: Inference snapshot

Implementation Note: "Impossible" Outcomes Depending on the way the conditional probability tables are generated, it is possible that, given a combination of certain evidence, some outcomes of a random variable may have the probability of 0.0. Mathematically correct, the software library SMILE that we use prevents setting conflicting evidence for such an outcome, pointing out that such an action is "impossible".

Nevertheless, we must be able to set such evidence for a number of reasons (e.g., the model may be incorrect, or the customer simply does not act rationally), because rejecting a customer-given answer as being "impossible" would be inacceptable for a sales dialogue. Hence, our practical implementation adapts the generated conditional probability tables to avoid probabilities of 0.0 by changing them to a very small positive value such as 0.000001 (subtracted from the greatest probability of the respective distribution). While this does not have a measurable effect on the calculations of the posteriori distributions (even less so if we use approximative algorithms), it keeps all outcomes "possible" and avoids this issue.

Note that calculating the conditional probability tables according to algorithm 4.1 avoids this issue altogether, since all elements are guaranteed to be $\neq 0$ (as they are obtained by multiplication and all factors are $\neq 0$ themselves). (Screenshots might show probabilities as "0%" due to rounding, however.)

6.3 Predictions

The first use of the inference engine is to give estimations about the BeliefState of ReasoningElements (as defined in section 3.3) about which the customer has not yet answered a question. These predictions will then be used to determine the relevance of a question according to definition 5.2.1 and can also be employed to adapt the question display as described in section 5.2.

Given a certain point in the dialogue (i.e., a certain amount of evidence set in the Bayesian network), we can obtain the belief state of a ReasoningElement from the calculated posterior probability distribution of the corresponding random variable. For a given ReasoningElement r, the BeliefState $\beta_r = \{(positive, \beta_r^+), (negative, \beta_r^-)\}$ (cf. definition 3.3.2) can be determined by calculating the degree of belief:

Definition 6.3.1 (Obtaining degrees of belief from the Bayes net) Let R denote a ReasoningElement, V its corresponding random variable and $dom(V)$ its value range. Let $posdom(V) \subset dom(V)$ and $negdom(V) \subset dom(V)$ be subsets of the value range, containing its "pos" and "neg" elements, respectively. Put differently, it holds that $posdom(V) \cup negdom(V) = dom(V) \setminus \{neutral\}$.

We define the *degrees of belief* β_r^+ and β_r^- as follows:

- $\beta_R^+ := \frac{\sum_{v \in posdom(V)} (i * P(V{=}v \mid ...))}{\sum_{v \in posdom(V)} (i * P(V{=}v \mid ...)) + \sum_{v \in negdom(V)} (i * P(V{=}v \mid ...))}$
- $\beta_R^- := \frac{\sum_{v \in negdom(V)} (i * P(V{=}v \mid ...))}{\sum_{v \in posdom(V)} (i * P(V{=}v \mid ...)) + \sum_{v \in negdom(V)} (i * P(V{=}v \mid ...))} = 1 - \beta_R^+$

One special case exists: Iff. $p(V = neutral \mid \ldots) = 1.0$ (i.e., evidence has been inserted for the "neutral" outcome), we define $\beta_R^+ = \beta_R^- = 0.0$, i.e., the system beliefs in *neither* of the two possible outcomes.

If desired, a simplification can be used to calculate the values of β_r for Traits or AttributeValues instead of definition 6.3.1. Since their random variables have binary value ranges, the BeliefState can be read directly from their corresponding conditional posterior probabilities, i.e.

- $\beta_R^+ := p(V = pos_1 \mid \ldots)$ and, correspondingly,
- $\beta_R^- := p(V = neg_1 \mid \ldots) = 1 - \beta_R^+$

6.4 Utility Estimations

We use a very simple way to estimate the utility of an AttributeValue for an Attribute. The BeliefState for an AttributeValue as introduced in definition 6.3.1 is closely linked to the posterior probability distribution of the corresponding random variable.

The random variable models the likelihood that a Product with the corresponding AttibuteValue us useful for the customer, given the knowledge available to the recommender system. The utility of that AttributeValue is therefore defined in a straightforward way:

Definition 6.4.1 (Utility of an AttributeValue) Let a denote an Attribute and $av \in dom(a)$ an AttributeValue. Further, let $\{(positive, \beta^+_{av}), (negative, \beta^-_{av})\}$ the calculated BeliefState of av. The *utility* $u_a(av)$ of av is then defined based on the degree of belief for *positive*:

$u_a(av) = \beta^+_{av}$

We can apply this definition to every AttributeValue in the domain model. The utility of the AttributeValues will then serve as basis for the recommendations as we will describe in section 7.1.

Example 6.4.2 (Utility of a complete Attribute) Consider the BeliefStates indicated by the state of the Bayesian network in example 6.2.1. For the Attribute "memory", the following utilities can be obtained:

$$\begin{aligned} u_{memory}(small) &= 0.0 \\ u_{memory}(medium) &= 0.75 \\ u_{memory}(large) &= 0.5 \end{aligned}$$

In other words, a medium memory size is suited best for the customer in this case.

6.5 Explanations

The inference engine can be employed in various ways to *explain* parts of the recommender system behaviour to a customer. These measures are useful to demonstrate understanding and, finally, to have the customer place trust in the generated recommendations.

We already encountered one of the explanation functionalities when discussing question relevance in section 5.2. By showing accurate predictions for the customer's answers, the recommender system demonstrates its competence. Although the chosen relevance measure means that questions suitable for this approach will commonly be only encountered late during the dialogue, the predictions can be used to make the reasoning behind the question order transparent to the user.

On explicit request, the predictions for other questions can be shown, making it clear that they are not considered very relevant at the moment due to their confident predictions. Furthermore, given our flexible dialogue model, the customer then has the simple option to switch to one of these questions to correct a false prediction directly.

Much in the same fashion, the utility estimations introduced in section 6.4 can be made transparent to the customer, allowing him/her to understand the reasoning behind an individual recommendation. When viewing a product datasheet, this information can also be used to highlight the "important" technical properties of a product in a personalized way.

The previous explanation techniques only relied on knowledge that was present in the Bayes network when performing its regular functions. Most interesting, however, is to use the diagnostic (i.e., "backwards") line of reasoning of a Bayes network to explain a given recommendation not in purely technical terms (i.e., which technical properties are important etc.) but instead using the terms that the customer used to describe himself/herself (i.e., in terms of the Traits and Needs that formed the basis of the dialogue so far).

In other words, given a Product with its concrete AttributeValues, we attempt to determine the Needs responsible for the recommendation. To this end, we set evidence in the Bayes net for the pos_1 outcome of each random variable corresponding to an AttributeValue of the current Product. We can then compute the posterior probability distributions for the "upper" layers of the Bayesian network, obtaining BeliefStates for all other ReasoningElements in the regular fashion.

This information may then be used to show to the customer the particular Needs a Product is especially useful for. By comparing these with the customer's answers, the information can be shown in a personalized way, i.e., which of the customer's preferences will be particularly satisfied by the Product and which wishes would remain unfulfilled.

It is noteworthy that the diagnostic view only depends on the Product at hand and is not itself personalized to the customer. As such, it may be precomputed and cached,

enabling instantaneous explanations for a large number of Products – which would not be possible if we had to do these calculations every time again because of the performance cost (cf. section 9.2).

6.6 Conclusion

In this chapter, we showed the design of the *inference engine* that provides all functions necessary for our recommender system. In particular, we showed how the higher-level requirements for such a component (as described in section 6.1) were implemented using the simple, probability-based interface of a Bayesian network generated according to section 4.2.

Most notably, the inference engine / Bayes network provides us with

- *Predictions* about future user bahaviour which are used for choosing the next questions and personalizing them;
- *Estimations* about the utility of technical properties which will be used to produce the actual product recommendations;
- *Explanations* about parts of the recommender system behaviour.

The Bayesian network obtains new knowledge by processing answers to questions as *evidence* for the outcomes of its random variables. This very flexible approach allows a mixed-initiative dialogue and easy belief revision.

Related Work

An approach similar to ours is presented in [JSSW95]. However, the utility estimations (the "value tree") of Jameson et al. do not seem to be built on an explicit model of the currently served customer but instead on the assumed properties of an average user of their system. Hence, the derived preferences are not personalized as strongly as in our approach. Also, as the value tree is a strictly hierarchical structure, it cannot capture the fact that a technical attribute may be influenced by more than one single need. Furthermore, it is not completely clear how informal statements (e.g., "I am a law student.") can be interpreted as relevant knowledge (e.g., an increased interest in politics) by the system apart from the possibility that a domain expert models this association directly within the Bayesian network.

The aforementioned approach of Pu et al [HP10, HP09] uses a matrix that directly links the elicited knowledge with probability distributions for the desired product properties. While we have doubts about the maintainability of such a structure (we will revisit this in chapter 8), we note the similarity of their technique to the specification of a full joint probability distribution of which a Bayesian network is a compact form. Apart from generating the recommendations, they do not use the matrix for dialogue management or explanation functionality.

In [MR07], an adaptive approach to select technical questions is presented. It is able to suggest "tightenings" (i.e. further questions) to reduce recommendation size based on a previously learned probability model. Their approach includes the possibility to re-learn the model when more dialogue histories are available. In contrast, our approach does not include a learning step but delegates that task to a domain expert. Also, our model is not concerned with direct connections between dialogue elements as is the case in [MR07] but instead specifies a more abstract view on the product domain from which the dialogue structure is inferred.

A domain model based on dynamic logic programming was introduced by Leite and Babini in [LB06]. Both customer and user model are represented using a large set of declarative rules which allows a detailed and powerful specification of the business domain – possibly even extended by user-supplied rules. However, the complex formal models appear expensive to maintain when confronted with domain changes. Furthermore, it seems unlikely that domain experts, much less customers, are able to express their knowledge by logic rules, whereas intuitiveness and maintainability of the model are two key points of our approach.

In [CL07] Cao and Li develop a recommender system for consumer electronics by using a fuzzy-based approach for the inference process which involves reasoning about a product's features. The approach does not include an explicit customer model and therefore is limited to reason only about the technical features of products. Therefore, its potential for a conversational recommender system that aims at complex product domains appears limited.

Bayesian networks are used by Ji et al. in [JSLZ03] to obtain recommendations in the commodities market. In contrast to our approach, they do not rely on a domain or customer model, but focus on learning the structure of the network and all probabilities from history data. Based on evidence provided by the current customer's purchases, other commodities are recommended depending on their posterior probabilities. This kind of evidence is not available in our application scenario, as the customer generally will make a single purchase and leave.

Park et al. use Bayesian networks for a very detailed user representation in [PHC07]. They use an expectation maximization algorithm to learn the conditional probability tables on their network. However, the structure of the network itself has been designed by a domain expert and is intended to remain fixed. Therefore, their approach requires extensive work when the underlying model changes.

7 Recommendation Generation

7.1 Utility Function

Our approach to generate recommendations is based on Multi-Attribute Utility-Theory ("MAUT"), as described in [WE86] and also in [RN10, ch. 16]. Basically, this technique is centered around the specification of a *utility function* that combines multiple properties of an item to calculate a score value, representing the "utility" (i.e., the degree of usefulness) of the item in question.

Common ways of applying MAUT (though often done informally) are the assignment of "school grades" to various aspects of an item which are then combined into some "average" grade and used to compare different items.

Other approaches compute fictional costs that convert arbitrary item-properties into monetary values to allow "fair" comparisons between items. A well-known application of the latter approach is the concept of the so-called "Total Cost of Ownership" [ES93].

The most important task with MAUT-based approaches is therefore to define an appropriate utility function.

For our scenario, the utiliy function must obviously set off the Attributes of any given Product against each other to condense their individual utility values calculated as described in section 6.4 into one single score value.

To this end, we use the widespread *weighted sum* approach that adds up each Attribute's utility value, taking its individual *importance* with respect to the recommendation process into account.

Thus, we can define the utility function for a given product in a general way (as originally published in [RF10]):

Definition 7.1.1 (Utility Function for an Article) Let A denote the set of Attributes of the Article in question and let p denote a corresponding Product.

We define the *utility* of p as

$utility(p) = \sum_{a \in A} (d_a \cdot i_a \cdot s_a \cdot u_a(av_a(p)))$, with

d_a the *distinctiveness* of a as will be defined in definition 7.1.2 below,
i_a the *importance* of a as in definition 7.1.4,
s_a the *situation factor* of a as in definition 7.1.5, and
$u_a(av_a(p))$ with $av_a(p) \in dom(a)$ the *utility* of the Product p for the customer with respect to a, as introduced in section 6.4.

The product $w_a = d_a \cdot i_a \cdot s_a$ is also called the *weight* of Attribute a. The three factors are based on the following considerations:

1. Those attributes that customers show significant interest in should be regarded as more important than others. In our model, "significant interest" is derived from the fact that more *distinctive* predictions for the attribute values exist.

2. A domain expert may assign a static numerical *importance* to each attribute (cf. section 3.2). Marketing research shows that some attributes are inherently more important than others in a buying decision. Our experiments indicate that it is sufficient to classify attributes into a small number of weight classes, which is considered to be simple for suitably knowledgeable experts.

3. The dialogue *situation* has influence on the importance of an attribute for the elicitation process. Attributes that are not connected to any already answered question, should not have any influence at all.

Definition 7.1.2 (Distinctiveness of an Attribute) The *distinctiveness* d_a of an Attribute a is defined as the average of the distances of the utilities of all possible attribute values $v \in dom(a)$ from the "indifferent" utility of 0.5. We multiply the fraction by 2 in order to normalize it to [0..1] (each term of the sum in the denominator yields a value in [0..0.5]).

$$d_a := 2\frac{\sum_{v \in dom(a)}(|u_a(v) - 0.5|)}{|dom(a)|}$$

As a sidenote, we would like to observe that the distinctiveness measure is closely related to the relevance of a question introduced in definition 5.2.1. If viewed in

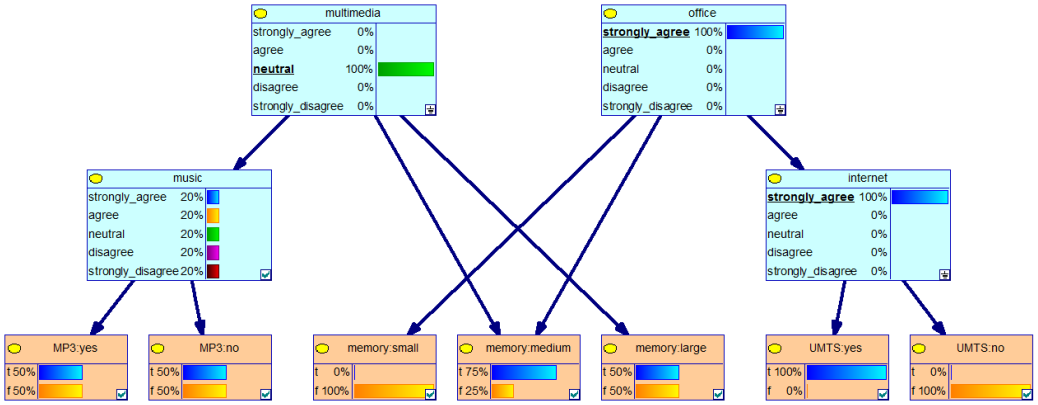

Figure 7.1: Inference snapshot *(Traits omitted)*

comparison, both measures have a very similar meaning, making it plausible that they are calculated in analogous ways.

Example 7.1.3 (Distinctiveness of an Attribute) To calculate the distinctiveness d_{memory} of the Attribute "memory", assume that the following utilities have been inferred (cf. figure 7.1):

$$\begin{aligned} u_{memory\_small} &= 0.0 \\ u_{memory\_medium} &= 0.75 \\ u_{memory\_large} &= 0.5 \end{aligned}$$

$$d_{memory} = 2 * \frac{|0.0-0.5|+|0.75-0.5|+|0.5-0.5|}{3} = 0.5$$

The result fits our intuition: We are not yet sure about the customer's opinion regarding memory size at this point of the dialogue. Therefore, values derived for this attribute should not be taken too seriously.

Definition 7.1.4 (Importance of an Attribute) The *importance* i_a of an Attribute a is a positive real number. It is commonly obtained from the current domain model as suggested by figure 3.3.

Definition 7.1.5 (Situation Factor of an Attribute) Let $Q_{answered}$ be the set of all questions already answered in the current dialogue. For an Attribute a let $P(a)$ denote the set of all Influence-ancestors of a, i.e., the Needs connected directly or transitively with a AttributeValue of a via an Influence-path in the network.

The *Situation Factor* s_a of a is defined as follows:

$$s_a := \begin{cases} 0 & \textit{if } \forall q \in Q_{answered} : q \notin P(a) \\ 1 & \textit{otherwise} \end{cases}$$

Example 7.1.6 (Weight of an Attribute) Extending example 7.1.3, we determine w_{memory} based on the following parameters for *distinctiveness*, *situation factor* and *importance*:

$$\begin{aligned} d_{memory} &= 0.5 \quad (\textit{example 7.1.3}) \\ s_{memory} &= 1.0 \quad (\textit{memory is connected to answered questions, Fig. 7.1}) \\ i_{memory} &= 2.0 \quad (\textit{taken from the domainmodel}) \end{aligned}$$

$$w_{memory} = 0.5 * 1.0 * 2.0 = 1.0$$

Notice how w_a combines i_a, a static, expert-defined value with d_a, a dynamic, customer-dependent value, taking these both important aspects into account. In particular, d_a must be updated after each dialogue step to account for the newly learnt knowledge. Of course, the same holds true for s_a, which brings the current status of the dialogue into the calculation.

Also, w_a would be a natural point to evolve the utility function, e.g., to include more factors, weigh factors differently, or similar extensions.

It is worth noting that, in general, the calculated utility for a Product does not have an absolute meaning (i.e., a statement about *how* useful the product is, cannot be made). The value can only be interpreted in a relative way, i.e., a higher utility means greater usefulness.

Example 7.1.7 (Preference Order by MAUT) Assume that the current product catalogue contains the entries as shown in table 7.1. The utility and distinctiveness

Name	**mp3**	**memory**	**umts**
MobileA	yes	medium	no
MobileB	yes	large	no
MobileC	no	medium	yes
MobileD	no	small	no

Table 7.1: Example product catalogue for the mobile phone domain

values are derived from the dialogue situation shown in figure 7.1:

$u_{umts\_yes} =$	1.0		
$u_{umts\_no} =$	0.0	$d_{umts} =$	1.0
$u_{memory\_small} =$	0.0		
$u_{memory\_medium} =$	0.75		
$u_{memory\_large} =$	0.5	$d_{memory} =$	0.5
$u_{mp3\_yes} =$	0.5		
$u_{mp3\_no} =$	0.5	$d_{mp3} =$	0.0

For simplicity, assume situation factors s_a and importances i_a of 1.0 for all attributes in this example. For this constellation, the following ranking of catalogue entries according to the utility values computed as shown above results:

1) MobileC (Utility: $1.0 \cdot 1.0 + 0.5 \cdot 0.75 + 0.0 \cdot 0.5 = 1.375$)
2) MobileA (Utility: $1.0 \cdot 0.0 + 0.5 \cdot 0.75 + 0.0 \cdot 0.5 = 0.375$)
3) MobileB (Utility: $1.0 \cdot 0.0 + 0.5 \cdot 0.5 + 0.0 \cdot 0.5 = 0.25$)
4) MobileD (Utility: $1.0 \cdot 0.0 + 0.5 \cdot 0.0 + 0.0 \cdot 0.5 = 0.0$)

Since UMTS capability is the most relevant feature for our sample business customer, it is decisive in producing the utility-based rank ("MobileC" is the only UMTS-capable device in table 7.1). In contrast, support for MP3 does not play a role since the course of the dialogue did not yet allow any conclusions about the customer's wishes in this respect.

7.2 Implementation in SQL

The utility function can be formulated as a standard SQL query. For the sake of simplicity, we assume that all relevant data for an article is available in a single table with one column for each technical attribute and an additional column containing the product's name. Every tuple in this table represents one concrete product (e.g., in the sample domain, one concrete mobile phone).

The following `SELECT` query computes a numerical `UTILITY` value for each tuple and orders the answer set accordingly:

```
SELECT   *, ($utilityfunction) AS UTILITY
FROM     article_table
ORDER BY UTILITY DESC
```

`$utilityfunction` calculates the overall utility of a given Article by summing up the utility of each attribute value u_{value} (as established in definition 7.1.1), weighted by $w_{attribute}$ for each Attribute:

```
$utilityfunction =
  $utility(att_1)$ * w(att_1) + ... +
  $utility(att_n)$ * w(att_n)
```

The utility of a single attribute is calculated using a set of SQL `CASE-THEN` clauses using the utilities of the attribute values u_{value} (cf. section 6.4):

```
$utility(att_x) =
  CASE WHEN att_x = val_x1 THEN u(val_x1) ELSE 0.0 END + ... +
  CASE WHEN att_x = val_xn THEN u(val_xn) ELSE 0.0 END
```

As an obvious optimisation, the `CASE-WHEN` clauses can be *nested* to enable early termination of the term's evaluation, which we do not show here for better legibility. Also, our actual implementation replaces the string-valued columns by numerical ones to avoid costly string comparisons during the query execution phase (taking advantage of the fact that all value ranges must be enumerated explicitly, cf. section 3.2). If the database supports `ENUM` column types, these can be used as another optimisation.

The structure of the query is known at the time of domain model design. Therefore, the query can be implemented as a stored procedure that can be called with the current weights and utility values as parameters. Since compiling the complex `CASE-WHEN` statements is a non-trivial task for the database server, this approach can provide a significant performance improvement.

In our experiments, the actual query execution times were only a few millliseconds (we will elaborate on performance measurements in section 9.2). It should be noted, however, that there are less than 1,000 different mobile phones on the market today, leading to a small product catalogue and thus a small domain size.

It is noteworthy that our queries do not limit the result set: It always contains *all* tuples of the corresponding Article's table with only the ordering dependent on the current situation, allowing a customer to examine the entire product catalogue. This is possible due to the generally small size of product catalogues in our use case and it is of course allowed to use a top-k operator to simply limit the result set size if desired.

Also, there are a few possibilities to place "hard" limits on the result set:

- When a customer has answered questions relating to Attributes, it is still possible that Products are ranked top that have one or more AttributeValues for which the customer has explicitly expressed a dislike, which may be irritating. Therefore, instead of providing input for the utility function, questions about Attributes can be used to create "filtering" options that are then treated as hard contraints (i.e., would be used to build a `WHERE` clause).
  - An alternative would be to modify the utility function to strongly penalize such Products, thus ensuring that they are never present in the top ranks.
- A hard `WHERE` clause could be built automatically if the *utility* of AttributeValues falls below a certain threshold (implying that Products with such AttributeValues are completely inacceptable for the customer).

However, it is clear that implementations must be prepared to deal with *empty result sets* whenever hard constraints are used. Our work does not focus on constraint / query relaxation techniques necessary to solve such an issue and we would like to argue that keeping the entire result set and allowing the user to choose freely from all alternatives (guided by the produced ordering) generally provides a superior option (see also [FFG+07, CP05] for the advantages of ordered recommendations).

7.3 Implementation Alternatives

The method described so far can also be used to provide the necessary inputs for more general approaches to rank query answers using Boolean predicates, such as the one presented in [BF08, BRF07]. To take full advantage of the more elaborate possibilites offered there, it would be necessary to extend our product modelling by ways to express the potentially hierarchical structure of complex boolean conditions.

Two more approaches were put into practice during the dissertation on an experimental basis:

7.3.1 Pareto-optimality Based Queries

The inference engine described above can be used to derive preferences for approaches based on *pareto-optimality* such as PreferenceSQL [KK02, Kie02]. In effect, the infer-

ence engine provides an ordering for all values of a technical attribute by interpreting the calculated utility values as the pareto-preferences serving as an input for PreferenceSQL.

The PreferenceSQL query language supports "`LAYERED`" preferences (cf. [Kie05]) for situations where a domain $dom(A)$ of an attribute can be partitioned into subsets that are ordered according to a "better than" relation. In our approach, all attribute values that have the same utility are grouped together in the same "layer", leading to a straight-forward application of the `LAYERED` preference constructor.

Clustering techniques may be used to limit the number of subsets that have to be considered by interpreting some values as equally preferred, despite minimal differences in their numerial utility values, in a roughly similar way to the discretization already applied to value ranges for Attributes.

Since the semantics of our approach relies on the notion that the customer is generally indifferent about attribute values with the same utility (i.e., all values with the same value are mutually substitutable), we annotate each `LAYERED` preference with the additional "`REGULAR`" keyword (cf. section 4 in [Kie05]). These preferences are combined using the pareto "`AND`" operator to form the complete PreferenceSQL query.

Example 7.3.1 (PreferenceSQL) We re-use the dialogue situation of example 7.1.7 to formulate a query in PreferenceSQL. Using the pattern described above leads to the following statement:

```
SELECT * FROM mobilephones PREFERRING
umts   LAYERED (('yes'), ('no'), others) REGULAR                    AND
memory LAYERED (('medium'), ('large'), ('small'), others) REGULAR AND
mp3    LAYERED (('yes', 'no'), others) REGULAR
```

Executing this statement against the product catalogue of table 7.1 yields "*MobileC*" as the query result, which is the pareto-optimal tuple of the relation and therefore the only result according to the "Best-Matches-Only" semantics of PreferenceSQL.

Successively re-executing the query with added `WHERE`-clauses to exclude the already-retrieved tuples yields the following results which are ordered exactly as those obtained using our MAUT-approach in example 7.1.7. Note that we do not claim that the orderings produced by both approaches are equivalent (in fact, generally they are not).

```
1)  MobileC
2)  MobileA  (WHERE name <> 'MobileC')
3)  MobileB  (  AND name <> 'MobileA')
4)  MobileD  (  AND name <> 'MobileB')
```

Generated in this straight-forward way, the PreferenceSQL query would consist of a very large number of pareto-preferences. To reduce the number of preferences, the weight of an Attribute may be exploited to limit the preferences to a fixed number or to consider only preferences for Attributes having a weight that exceeds a certain threshold.

7.3.2 NoSQL

The possibility of implementing the recommendation functionality on the basis of recent NoSQL-techniques (cf. [EFHB10]) was investigated in the course of a bachelor's thesis at our chair [Kar11], using the document-based database "CouchDB" (cf. [ALS10]).

Conceptually, using this approach, Products are seen as *documents*, i.e., the largely schema-less entities that form the primary data objects of CouchDB as contrary to tuples. The utility function as in section 7.1 remains but is now used as a mapping function for the Map/Reduce-based querying approach of CouchDB (cf. [DG04]). The result of auch a query is an ordered map, containing all Products indexed by their utility value as key.

However, while the idea is interesting, the personalized nature of our utility function prevents CouchDB from using its internal caching abilities (a mechanism very similar to materialized views in RDBMS), resulting in a very poor runtime performance (i.e., several orders of magnitude worse than the plain SQL approach). An acceptable performance – but still significantly slower than all other approaches – could only be achieved by using extensive application-side caching.

A possible advantage of this approach is the integrated distributability of queries over several database nodes. We could envision manufacturers to provide the technical datasheets of their products in a CouchDB instance, enabling, among other things, querying from our recommender system. However, we remain doubtful whether the performance issues can be alleviated in the foreseeable future, since our use-case (i.e., a constantly changing map function) does not appear to be of primary concern.

7.4 Conclusion

In this chapter, we showed how our recommender system generates recommendations by sorting the entire product database using a utility function.

As an application of Multi-Attribute Utility-Theory, the utility function uses a weighted sum approach to add up the utilities of the individual AttributeValues of a given Product to form a single value representing a relative indicator of the usefulness of the product. Let us stress again that the utility value is only meaningful in comparison to other values calculated from the exact same utility function. It does not provide an absolute measure or can be sensibly compared with utility values corresponding to a different state of the dialogue.

All information necessary to construct the ranking function is either available in the domain model or can be obtained from the inference engine. We then showed how the utility function is implemented in plain SQL and mentioned alternatives using ranking- or pareto-based database querying techniques, as well as an experimental NoSQL implementation.

Related Work

PreferenceSQL by Kießling et al [KK02, Kie02] was already mentioned in section 7.3.1 and is quite compatible to our MAUT implementation. In fact, we implemented a PreferenceSQL connector that can be used in our software based on a simple configuration switch, subject to the following observations. Since both approaches use different semantics, i.e., Best-Matches-Only vs. MAUT, the query result are generally not equivalent, however.

Also, when used naïvely, our approach would generate a large number of pareto-preferences (i.e., one per Attribute) which practically guarantees that no single Product fulfills all, or even most, of the preferences. Consequently, PreferenceSQL successively relaxes the number of preferences considered to check whether that layer contains pareto-dominant items. I.e., given n preferences, it tries all combinations of *(n-1)*–of–n preferences, then *(n-2)*–of–n and so on, until dominant items are found.

However, since the number of these combinations grows quickly, this approach has a somewhat negative impact on performance and, worse, frequently leads to a situation where a given layer did not yield results yet, whereas the successive layer provides very many results (up to the entire catalogue, in some observed cases). While this is not

a problem per se (our approach always returns the entire dialogue), the result set of PreferenceSQL contains *only* dominant items. Since, based on the pareto-semantics, no sensible ordering can be defined over dominant items, the usefulness of this approach is very limited for our use case. It is a necessity to strictly limit the used number of preferences and our experiments indicate plausibly useful result set sizes only with as few as 5-10 preferences, which means that a large amount of knowledge would be ignored.

On the other hand, should PreferenceSQL find, e.g., a single dominant item, its Best-Matches-Only semantics would return only this item, making it impossible to compare the recommendation to runners-up or similar actions.

As another related approach, Ardissono et al [AFF+02, AFF+03] use their recommendation dialogue to gather a number of *constraints* relating to the technical properties of the products they have to recommend. These constraints are then used by a general constraint-solving application to look for those elements of the product catalogue that form *solutions* of the posed constraint-system. Unlike our sorting-based approach, they generally have to deal with the fact that the constraint-system may be un-solvable, i.e., there are no products in the catalogue satisfying all requirements. For this case, they employ automatic constraint-relaxation techniques that subsequently drop constraints until the system can be solved again. In this, their approach is quite similar to PreferenceSQL's behaviour mentioned above, although they attempt to drop constraints based on their perceived importance, comparable to our notion of *weights* for an Attribute.

8 System Lifecycle

Knowledge-based recommender systems are typically known to require extensive maintenance efforts, particularly for dynamic business domains like our primary application scenario (cf. section 2.4.1). Hence, in order to provide an adequate solution for our use case, we specifically took maintainability issues into account during the design of our approach.

Our considerations led to the layered, metamodel-based, generative design that has been described so far. It is built to support short- and medium-term maintenance scenarios efficiently, or even automatically, while being flexible enough to allow an evolution of the developed ideas in the longer term.

8.1 Short Term: Instance Modifications

The most frequent maintenance scenario concerns changes in the domain model instance, e.g., the appearance of new Products or changes regarding some AttributeValues (such as "price"), that can, however, still be expressed in terms of the current domain model.

We anticipate that such updates occur between several times a month and daily. For example, in our mobile phone use case, new phones are commonly released twice a month with the option of doing weekly price updates in between. In our movie scenario, new DVDs / Blurays are released on a weekly basis.

It is noteworthy that recommender systems based on collaborative filtering are completely unable to handle this situation without requiring significant effort. Since new products have not yet been bought or rated by anyone, they are completely invisible to the buyer profiles and, consequently, are never recommended. To solve this issue, such systems have to combine their collaborative approach with item-based techniques that allow to transfer ratings between items based on their similarity. This way, a new item will "inherit" the rating of existing items that are comparable to the new product.

Contrastingly, in our approach, new and changed Products are integrated into the recommendation process *transparently*. Once present in the product catalogue database, the utility function can calculate their utility in exactly the same way as for older Products, enabling a seamless integration of doman model instance changes into the recommendation process. In fact, there is no conceptual obstacle to allowing this even *during* a running recommendation dialogue.

The feasibility of the update process was verified during the cooperation with our industry partner. There, our prototypical implementation regularly synchronised its internal database with a centrally managed product catalogue that was continually updated during the day-to-day operations of the industry partner. Also, the product catalogue for our movies recommender system was manually updated several times to keep up-to-date with new movie releases.

However, our project experiences also demonstrated the limits of automation in this respect. At first, it was attempted to acquire product datasheets directly from the manufacturers or third parties, and to import these into the recommender system in a fully automated way. There exist commercial service providers that offer product datafiles for an extremely wide range of article classes on a subscription basis, aggregating the data published by the individual manufacturers. The data sheets are available in easily machine-readable formats, e.g., as Access-databases or CSV-textfiles.

Automatically importing these datafiles required significant effort, however. The data vendor did not provide compulsory schema information, leading to a large number of synonyms, i.e., different representations of the same information, such as “4 Megapixels”, “4.0 MP”, the integer “4”, or combinations thereof, including typing errors, for digital camera resolution. While it proved feasible to normalise a given set of input data, the lack of schema implied the risk of encountering previously unknown synonyms with every new data set, limiting the trust that could be placed onto an automated import engine. Also, none of the data fields was guaranteed to be present, which leads to potentially incomplete product descriptions. While the product model is principally able to handle incomplete data as described in section 3.4, doing so is detrimental to the recommendation quality.

Finally, the inclusion of new products into the delivered data files always lagged behind the release dates by a certain time span. Since this directly opposes our goal of “instantaneous” recommender system updates, this was the primary reason for our industry partner to build the in-house managed product database mentioned above, along with the data quality issues, which were largely eliminated in the process.

8.2 Medium Term: Model Adaptations

In our approach, the domain model codifies the product structure and the marketing knowledge for a given domain. Both are subject to change, though not as frequently as the domain instance as we described in the previous section. Maintaining a recommender system's knowledge base is generally considered to be an expensive task and our metamodelling approach was specifically designed to keep the maintenance requirements low.

8.2.1 Technological Changes

The first need to adapt the domain model results from technical innovation in the domain that cannot be adequately represented in the current model. In particular, this means that the utility function (cf. section 7.1) is generally unaware of these new technical properties and consequently fails to take them into account.

Generally, two categories of technical changes can be distinguished: Adding a new Attribute, and extending the value range of an existing Attribute by new AttributeValues. It is similarly imaginable to *remove* Attributes or AttributeValues, but in practice there is no reason to delete such knowledge.

For example, in our use case, the pervasiveness of GPS receivers is a relatively recent development, requiring integration into the recommendation process – even more so if we consider that the development sparked new services along with it that, again, had an influence on technical properties like a requirement for online connectivity. As an example for the need to extend a value range, consider the ever-increasing megapixel numbers for the integrated digital cameras – just recently a mobile phone with a resolution of 41 megapixels was announced[1].

In the most simple case, the new AttributeValues can be integrated into the domain model by connecting them to existing InfluenceSources (i.e., Traits and Needs) via appropriate Influences. Should it be known that the new element of the value range is essentially "better" than the existing AttributeValues, a new Influence could be suggested more or less automatically.

In other cases, like the aforementioned GPS receiver, the existing questions may be insufficient to elicit the knowledge necessary to establish whether a particular technical function is desirable for the customer. Here, it would be necessary to extend the

[1] see `http://www.heise.de/-1443641.html` – in German, last accessed on 05.03.2012

customer model by additional Needs, leading to new questions in the dialogue. Of course, new AttributeValues may be connected to a mixture of old and newly created Needs in the customer model.

The fact that Influences represent only simple causal relationships between a single source and a single target is particularly helpful in keeping the necessary model adaptations local and intuitive. The Bayesian inference engine described in chapter 6 ensures that the entire model remains consistent, i.e., by propagating evidence along the modified Influences and adapting its conditional probability tables accordingly.

8.2.2 Marketing Changes

In addition to adapting the recommendation dialogue to technological changes in the domain, new marketing knowledge may be found independently and may need to be integrated into the recommendation process.

As an example from our use case, *senior citizens* were recognized as a potential buyer group for mobile phones only rather recently. In order to take these customers into consideration, new Needs and Traits/TraitGroups would have to be created (example 3.3.7 already contains a corresponding age group), along with the appropriate Influences to connect the new ReasoningElements to the rest of the model.

Other, simpler examples of modifications that are more likely to occur during the day-to-day maintenance of the recommender system are the addition and removal of a few Influences, or the optimisation of some weights.

In many cases, the product model can remain untouched in spite of such customer model adaptations. Then, the only step necessary to incorporate the new marketing knowledge is to re-generate the Bayesian network. However, reciprocal to technological changes, new ways of marketing may also necessitate the gathering of previously disregarded technical properties. For example, when targeting senior citizens, larger keypads are a property that is related to their needs. Note that, of course, these properties have always been present in the product, they may just not have been registered in the product model.

For this reason, it proved useful during the course of our practical project to consider all available technical information for inclusion into the product model, independent of whether a certain Attribute was currently relevant for the concrete recommendation process. Both the Bayes network generation and the utility function can be trivially optimised to ignore isolated Attributes (i.e., Attributes whose AttributeValues are not

connected to any Influences at all), practically eliminating any performance issues that might arise from an "unnecessarily rich" product model.

In fact, our industry partner took the opportunity to build a very general product catalogue system that contains *all* available technical information about the product they sell. As mentioned in the previous section, such a database of high quality product information has a value in itself and provisioning our recommender system with a subset of its data is only one of its intended usage scenarios.

8.2.3 Model Editor Application

While the domain model is already designed to hide the complex mathematical formalisms from the user, it is necessary to provide further support in the form of a convenient editing application that can be used to solve the model maintenance scenarios described above.

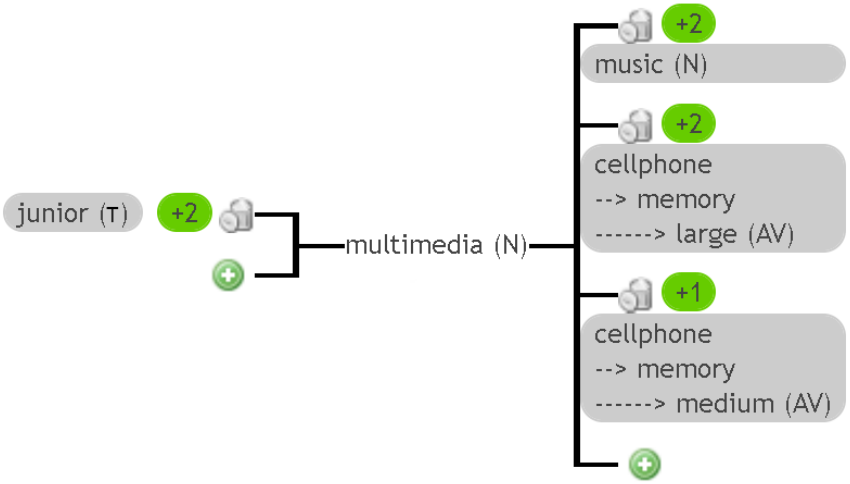

Figure 8.1: Domain model editor screenshot

We have built a prototypical domain model editor centered on the idea that the graph that represents the domain model is too large to be comprehended if presented in its entirety (cf. figure 3.8). Hence, we decided to provide partial views on the graph, always visualizing a single ReasoningElement along with its parents and children with an easy navigation along the edges through the graph (see figure 8.1 for a screenshot). This approach allows a domain expert to work along two principal paradigms:

- The "top down" view, i.e., which ReasoningElements *are influenced by* the model element currently under consideration?
- The "bottom up" view, i.e., which ReasoningElements *influence* the current model element?

Of course, both views may be mixed during editing, depending on individual preferences.

In addition, the editor supports the import and export of spreadsheet representations of the domain model (in the form of an adjacency matrix representing the Influences as shown in figure 8.2) and the Product data (see figure 8.3). We have found that using the spreadsheet format is suitable and efficient for the initial creation of a customer model, whereas the graph view is preferable for the minor modifications common in the day-to-day operation.

			Age	Age	Age	Need	Need	Need	Need
			junior	middle	senior	multimedia	music	office	internet
	Need	multimedia	2	0	0	0	0	0	0
	Need	music	0	0	0	2	0	0	0
	Need	office	0	2	0	0	0	0	0
	Need	internet	0	0	0	0	0	2	0
cellphone	mp3	yes	0	0	0	0	2	0	0
cellphone	mp3	no	0	0	0	0	-2	0	0
cellphone	memory	small	0	0	0	0	0	-2	0
cellphone	memory	medium	0	0	0	1	0	1	0
cellphone	memory	large	0	0	0	2	0	0	0
cellphone	umts	yes	0	0	0	0	0	0	2
cellphone	umts	no	0	0	0	0	0	0	-2

Figure 8.2: Influences adjacency matrix spreadsheet

	A	B	C	D
1	Productname	mp3	memory	umts
2	MobileA	yes	medium	no
3	MobileB	yes	large	no
4	MobileC	no	medium	yes
5	MobileD	no	small	no

Figure 8.3: Product catalogue spreadsheet

The editor application is integrated into the software landscape of our industry partner and used during the regular system maintenance operations there (in conjunction with

keeping the aforementioned product catalogue up-to-date). In fact, the domain model for our primary use case was revised several times during the course of our project – partly using the aforementioned spreadsheets and partly using the editor application.

8.3 Long Term: Evolution of the Metamodel

Although we have of course designed our metamodel to fulfill the current requirements of our primary use case, we have taken care to keep the metamodel very general and to not include elements that would be overly specific or restricting to a particular marketing domain. Still, we cannot assume that our metamodel will remain immutable forever, particularly if we consider a possible desire by users to include those very domain-specific extensions/optimisations that we explicitly avoided in our more general approach.

With regard to this anticipated evolution, our system design introduces a separation of concerns between the metamodel and the actual implementation via its generative approach. As long as the intermediate layer of our architecture remains constant, such changes require only adaptations to the generation function that transforms an instance of the metamodel into the intermediate representation. The handling of the intermediate representation itself remains untouched, keeping the required software changes local.

As an example, consider the possibility of allowing Influences to/from "neutral" BeliefStates that we mentioned in section 4.1.2. The intermediate representation already implicitly allows for the corresponding edges, making the extension possible without having to touch the generation algorithm for the Bayes network or the Bayes network usage.

Another example is more concrete, coming from our project use case. When investigating how to extend the recommendation capabilities from mobile phones to encompass the corresponding *rate plans*, we found that the predominant criterion for customers to choose their rate plan is, not surprisingly, the *price*. However, the actual cost of a rate plan usually depends on the customer's behaviour and can be surprisingly difficult to calculate even when the behaviour is known, due to extremely varying and complicated billing schemes, according to our industry partner, therefore an exhaustive treatment of this subject is beyond the scope of this dissertation.

To integrate the necessary knowledge into our approach, the metamodel had to be extended by adding approximations for a customers behaviour regarding dimensions

like “call minutes per month”, “text messages sent per month”, or “megabytes downloaded per month”. As a first step into that direction, we extended the Traits to carry corresponding data fields as illustrated by the UML diagram in figure 8.4. During the recommendation dialogue, this information is then provided to a rate calculator that returns a price approximation for the available rate plans. The approximation then serves as an additional input for the product ranking in addition to the utility function described in section 7.1.

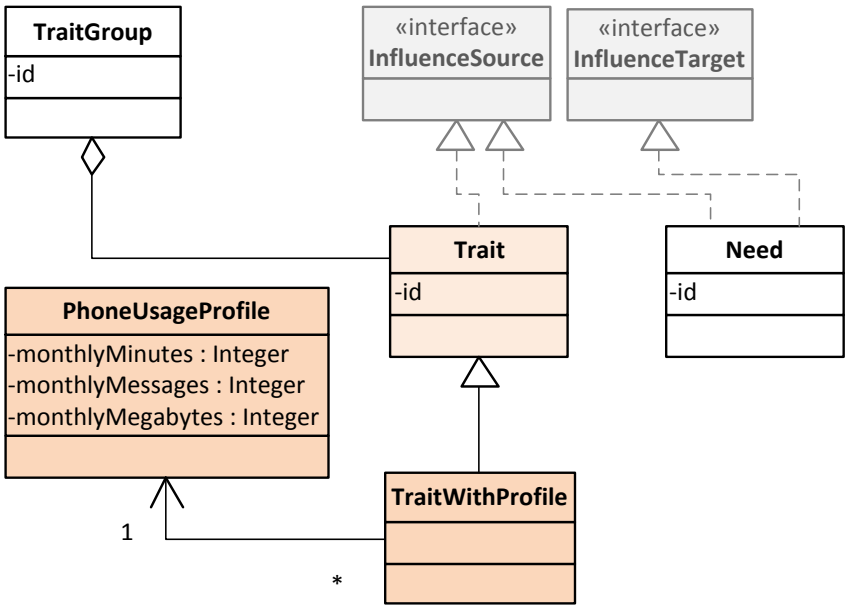

Figure 8.4: UML diagram of a metamodel extension

One more example from our project experience does not concern an extension of the metamodel per se, but primarily a modification of the intermediate model generation algorithm, changing the interpretation of how Influences operate.

Given an Influence $I = (S, positive, T, positive, w)$ (without loss of generality), the current algorithm creates a number of edges that connect the “$pos_?$” elements of the value ranges of the variables S and T (cf. section 4.1.2). An earlier version of the algorithm created additional edges equivalent to those that the current algorithm would create if we assumed an additionally present Influence $I' = (S, negative, S, negative, w)$.

In other words, the older interpretation of an Influence contained an “inverse” semantic, implying that if the presence of a particular feature fulfilled a given customer need,

the absence of the need would also make the absence of the feature more desirable than its presence. We found that this did not adequately capture reality as illustrated by the following example:

Example 8.3.1 Consider our domain model from example 3.3.7 that contains an Influence connecting the Need "music" to the AttributeValue "MP3: available" with $(cause = positive, effect = positive, weight = 2)$.

Using the old semantics, not having the Need "music" would have meant that mobile phones *without* MP3 players would have been preferable to mobile phones that have MP3 functionality. In reality, however, a customer without particular wishes regarding music would be *indifferent* towards MP3 functionality and the new semantics reflects this.

Note that the former semantics can always be achieved by explicitly adding a corresponding "inverse" Influence.

This modification was achieved by adapting the generation algorithm of the intermediate representation only. In particular, neither the treatment of the intermediate model – once generated – nor the metamodel itself or any instances of it needed to be modified.

8.4 Bayesian Network Construction

The multi-layered design approach serves to "protect" the internal implementation from changes at the domain model and metamodel layers as shown in the preceding sections and, vice versa, ensures that the implementation may be modified without implications for existing models. Obviously, this property was at its most useful during the development phase before the implementation was stable, but it still provides the opportunity to adapt the implementation or to investigate technology alternatives as demonstrated by the experimental inclusion of PreferenceSQL and NoSQL techniques for the recommender engine (cf. section 7.3).

We would like to elaborate on another significant modification in the construction of the Bayes network that illustrates our improved experience for the recommendation processes during the course of our industry project. Due to the abstraction provided by the layered design, the modification could be executed without requiring adaptation to our existing domain models, leading to an "instant" improvement of recommendation quality.

Calculating the conditional probability tables ("CPTs") of the Bayes network in the way described in section 4.2 supersedes an earlier approach, originally published in [RKF08]. If we call the current algorithm "OR-based", the calculation of [RKF08] can be described as "AND-based". In other words, whereas currently *one* cause is sufficient to achieve an effect with sufficient certainty, the older algorithm drew the likelihood for the effect based on a *weighted sum* of the number of *all* fulfilled sources.

An OR-based CPT is constructed by using a *product* of its parents' individual inhibition probabilities, leading to the overall inhibition probability (cf. proposition 4.2.7):

$$P(T = t|S_1 = s_1, \ldots, S_n = s_n) = \frac{1 - 0.5(\prod_{e \in E_t} ipc(e.w))}{\sum_{E_t \in E_T} (1 - 0.5(\prod_{e \in E_t} ipc(e.w)))} \tag{8.1}$$

In contrast, an AND-based CPT can be roughly characterized by a *sum* of its parent Influences weights (cf. [RKF08]):

$$P(T = t|S_1 = s_1, \ldots, S_n = s_n) = \frac{\sum_{e \in E_t} e.w}{\sum_{E_t \in E_T} \sum_{e \in E_t} e.w} \tag{8.2}$$

It is noteworthy that both approaches conform to the semantics of an Influence as in definition 3.3.4. Yet, as equation 8.2 shows for the AND-based approach, a great likelihood for a particular outcome can only be achieved if E_t is a large subset of E_T. This is generally only the case if the valuation $S_1 = s_1, \ldots, S_n = s_n$ corresponds to many "fulfilled" Influences, which does not adequately capture reality for our use case, as the following situation illustrates:

Some AttributeValues, particularly those relating to some mobile phone manufacturers have a large number of incoming Influences, resulting from the fact that the manufacturer represents a certain *brand image* that is supposed to be desirable for people in many situations. In this case, the Influences (or, more precisely, their *sources*) can be regarded as *reasons* to buy a mobile phone of the particular brand. Conversely, adding a new parent / reason leads to a general *decrease* of the likelihoods for many valuations of that variable in our Bayesian network, which is clearly against the intention of the model designer. The new OR-based approach remedies this, as a new reason can only increase the likelihood (i.e., by reducing the remaining inhibition probability).

8.5 Conclusion

In this chapter, we showed how our recomender system supports changes, both in the targeted market domain(s) and the underlying algorithms and the metamodel. We provided evidence of the proposed capabilities by giving examples from our project experience where the system is deployed and actively maintained at all described frequency levels.

In particular, short term maintenance can be largely automated (provided a high-quality data source exists) and medium term model maintenance is efficiently supported by a dedicated editor application. Also, the layered system architecture provides the opportunity to extend the metamodel or to adapt the underlying algorithms. These are properties that we put to good use during the project phase of this dissertation.

Related Work

The conversational recommender system described in [AFF+03, AFF+02] follows a similar approach to ours for the product model and should basically be able to achieve comparable performance for short term maintenance (i.e., for the integration of newly released products). However, their explicit dialogue specification appears more expensive to maintain in the face of changes within the domain. To incorporate new marketing knowledge, the entire graph formed by the dialogue steps would have to be analyzed to find the suitable places where to integrate new questions into the several dialogue paths.

The questionnaire-based recommender system in [HP10, HP09] uses a matrix to provide a direct mapping between the personality traits derived from the questionnaire and the product properties. The matrix is obtained from a user survey. Since they use a standard personality quiz, there is generally no need to integrate new questions. However, it remains unclear how new marketing knowledge can be taken into account apart from continually renewed user surveys. In addition, it remains unclear to us, how the approach would scale to more complex product models that are described by more than a single product attribute due to the significant growth of the matching matrix.

9 Evaluation

9.1 Practical Adequacy of the Approach

Recommender systems are generally found to be difficult to evaluate because it is practically impossible to measure the direct impact of the recommendations [JZFF11, ch. 7]. Furthermore, for conversational recommenders, it is practically impossible to find a system that is comparable enough to serve as a benchmark. In addition, with knowledge-based approaches, it is always hard to separate the quality of the knowledge model from the quality of the approach itself. During our project, our industry partner constantly refined the used domain model until we were reasonably confident that it was sound and complete to a reasonable degree – implying that, should the recommender system commit mistakes, these mistakes would have to be found within *its interpretation* of the model.

The best imaginable evaluation would have been to measure the effect of using the recommender system in a real-life application (in terms of increased revenue or the like). Unfortunately, this was not possible within the limited time frame of our industry cooperation project and we sought another way to judge the developed approach.

Our industry partner employed an independent market research institute to conduct a study [Cen10, Cen09] to evaluate the quality of the recommendations provided by the implementation of the recommender system. While we cannot claim credit for conducting the study ourselves, we would like to report about its methodology and results.

The institute began its research by defining nine *clusters* of customers with similar properties [Cen09]. Each cluster was described in terms of attitudes and desires, particularly with respect to the mobile telecommunications domain. For each of the clusters, a dedicated scoring model was created, by using the product model of our prototype (i.e., the Article definition with its Attributes and AttributeValues) and deciding about the relative importance of the Attributes and the suitability of certain AttributeValues for a representative member of each cluster.

It should be noted that the institute, at this point of the study, was deliberately unaware of the other parts of our domain model, most notably the Needs and Influences. Also, they were not informed about the internal workings of our approach, in particular about our use of a Bayesian network and a MAUT-based ranking approach. Therefore, it can be confidently said that their modelling was conceptually independent of our domain model and approach.

Interestingly though, their chosen approach was very similar to our own, since, in fact, they created a utility function not unlike the one described in section 7.1. Whereas their scoring model / utility function is manually defined for each cluster, our system uses the Bayesian network to derive the function individually and automatically for each customer. The scoring model was then used to obtain one ranked list of the available mobile phones for each cluster which would serve as a benchmark for the recommendations provided by our approach.

To be able to obtain recommendations from our system, the institute was provided with the list of questions that were represented in our domain model at the time of the study. By relying on the attidues defined for the clusters, the institute answered the questions from the point of view of a representative customer from each cluster.

These pre-defined answers were then used in interactions with our prototype to record the provided recommendations at two points in the dialogue: after answering 10 questions and after completing the entire dialogue.

To estimate the similarity between the recommendations from our system for a given customer and the respective benchmark ranking, the distance (in positions) between each Product in the Top-20 recommendations was calculated and graded. Accumulated over all clusters, the following results were found as illustrated by figure 9.1 (taken from [Cen10]):

- More than 95% of the recommendations were considered at least “good”, based on their distance in the two ranked recommendation lists.
- About one third of the recommendations was considered “excellent”, meaning that they were either at the same position in both rankings or only one position apart.
- The differences in quality between the clusters were not found to be significant for practical use.
- During the dialogue, the recommendation quality improves and even compensates intentional wrong answers during the first ten questions.

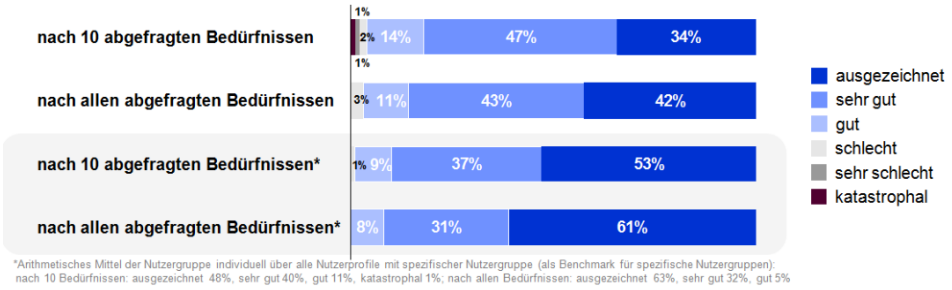

Figure 9.1: Study results, showing the distribution of grades accumulated over all customer groups and for the "individual" cluster in particular

Another, not as rigorous evaluation comes from using our prototype application in the context of university marketing events, such as Girls' Days or study information days. There, we organized workshops in which the participants, commonly eigth to tenth graders, form teams of four or five and have to use the editor application to build a simple domain model for a simplified movies domain (cf. section 2.4.2) – for which we assume a general level of knowledge with the participants.

The timeframe for the workshops is three, possibly up to four hours and includes an introduction into the modelling techniques to use, defining Needs and Influences and actually trying out the recommender system based on the built domain model. We can report the following results from these workshops:

- Despite the minimal introduction, all participants were able to understand the modelling techniques and use them to build a domain model in their teams. All of the about 20 teams that participated until now (except for one group with technical difficulties not directly related to the task) successfully completed the task within the time frame, i.e., they built a complete domain model and were able to try it out.
- Despite their simpleness and a significant variance in the taken approaches, the recommendations produced by these models were generally plausible and comparable in quality to the "reference" model that was built by the members of our chair in preparation of the prototype.

These experiences are admittedly limited but we argue that they nevertheless allow to conclude that our domain metamodel fulfils its principal requirement of being easy enough to use for domain experts. Furthermore, they demonstrate the general transferability of our approach beyond the mobile telecommunications use case.

9.2 Performance

9.2.1 Runtime Measurements

The principal requirement for our approach is its applicability in the context of a web application or a "kiosk" style PC. In other words, the recommender system must be responsive enough to be used in an interactive way, allowing for a maximum delay between request and answer of, say, a few hundred milliseconds.

Our in-house measurements with our prototype were conducted on one of our standard lab PCs with the following technical configuration. Unless specifically mentioned, all measurements were conducted on the basis of the real world use case as described in example 3.4.1.

CPU	Intel Core2Duo @ 2.13 GHz
RAM	2GB
Operating System	Microsoft Windows 7 Professional
Application Server	Tomcat 7
Database Server	MySQL 5

9.2.1.1 Bayesian Inference

Executing the inference with our Bayes network is the primary cost factor with respect to runtime. Exact inference, using the standard "clustering" algorithm, is infeasible in our real-world use case with regard to both time and memory constraints. Hence, we decided to use approximative "sampling" algorithms (cf. section 2.2) which are generally less demanding in memory consumption, requiring no significant amount of memory beyond the necessity of storing the Bayes network itself. Also, they possess the innate ability of balancing inference quality against runtime by varying the number of samples used for the approximation.

In the course of a master's thesis [Alt09], the different sampling algorithms provided by the SMILE library were investigated regarding their suitability for our specific use case. Surprisingly, while the "advanced" algorithms like AIS or EPIS (cf. section 2.2.4) would be considered a generally optimal choice, the simpler algorithms Likelihood Weighting and Heuristic Importance Sampling proved superior with regards to performance, while not incurring a loss in estimation quality. Our implementation can be configured to use any available algorithm, but consequently defaults to Likelihood

Weighting. The study results in section 9.1 are based on this algorithm with a sample count of 8,000 (as will be detailed in section 9.2.1.4 below).

We have measured the execution time of the "update beliefs" request to obtain the posterior probability distributions after setting the answers, by repeatedly executing the "cluster" dialogues described in section 9.1. Overall, each of the 9 dialogues consisted of 23 individual steps (one TraitGroup and 22 Needs in the model) and was executed 10 times, resulting in 2070 measurements which form a representative sample of typical workloads expected for using the recommender system with varied portions of the random variables set as evidence. Figure 9.2 shows our measurement results, displaying a roughly linear development for increasing numbers of samples.

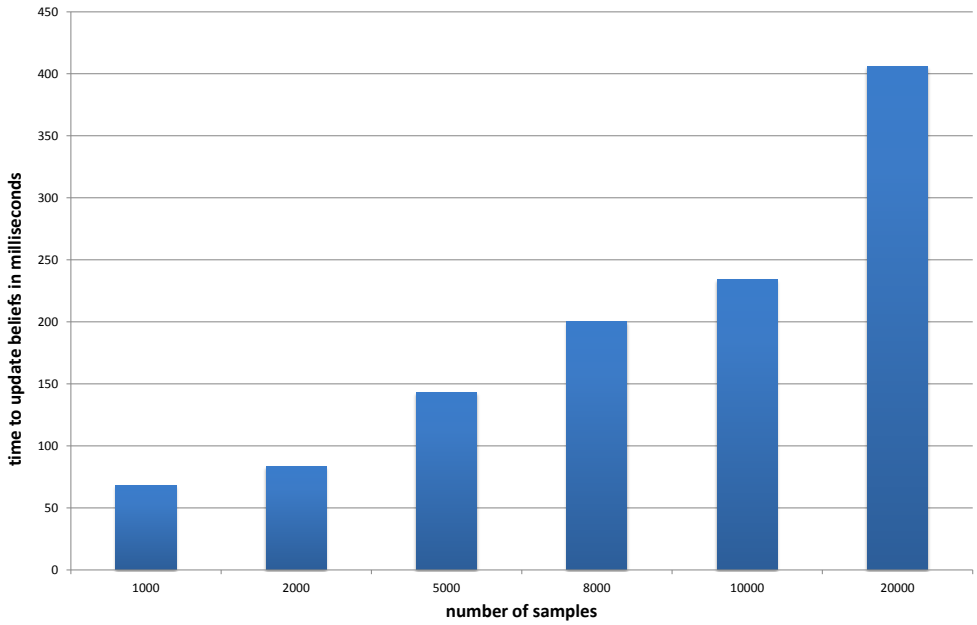

Figure 9.2: Bayesian inference performance measurements, showing median values

9.2.1.2 Database Queries

We have measured the execution time of the database queries to obtain the recommendation list, by using the same experimental setup as described in section 9.2.1.1 above. Again we believe that the more than 2000 measurements form a representative workload for our system.

Our measurements compare our own utility-based approach with the corresponding queries in PreferenceSQL (cf. section 7.3.1), as shown in figure 9.3. In addition to the "full" queries considering all 23 Attributes, one set of tests was run in which only the 5 most important Attributes were included in the pareto conditions / utility function. This was done in order to create a more "favourable" workload for PreferenceSQL, which performs very poorly when faced with a large number of preferences, as we will describe below.

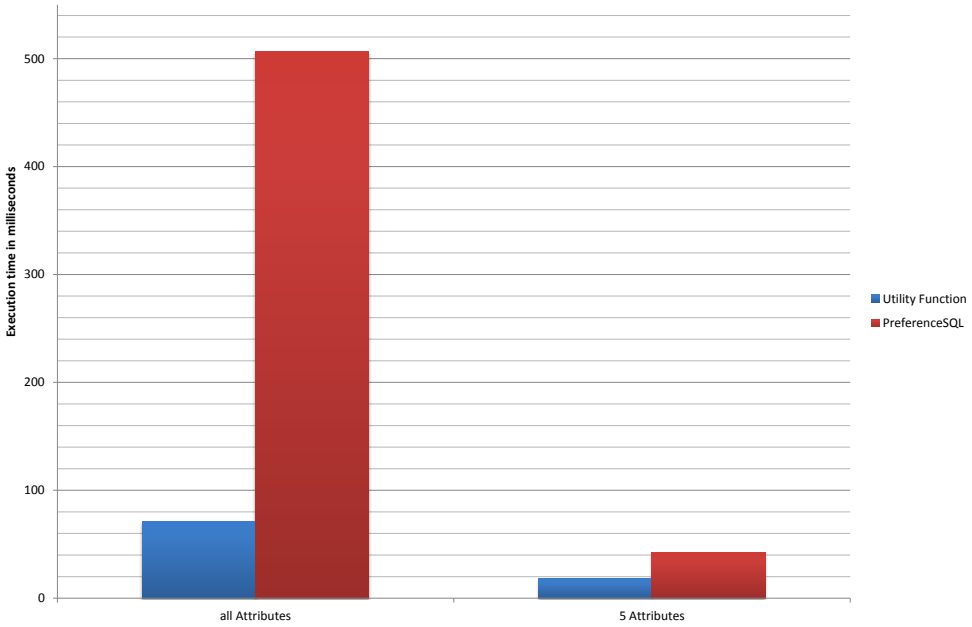

Figure 9.3: Database performance measurement results, showing the median execution time of recommendation queries in milliseconds

The measurements show the following results:

- The utility-based database queries (cf. section 7.2) complete in less than 100ms in all cases, which is adequate for our requirements.
- When all Attributes are considered, PreferenceSQL must be regarded as too slow, with requiring more than 500ms to provide recommendations.
- Both approaches take advantage of the reduced query complexity that results from lowering the number of evaluated Attributes. For PreferenceSQL, a reduction to 5 Attributes yields adequate performance for practical use.

To understand the speed deficiency of PreferenceSQL in our use case, one must consider how its pareto-based semantics are implemented: Given a set of n preferences, the database is queried for items that are pareto-optimal for all n preferences. If there are none, the database is queried for a union of all pareto-optimal items given each *(n-1)*–of–n subset of the preferences set. This is repeated for all *(n-2)*–of–n subsets, *(n-3)*–of–n subsets, and so on, until one of the queries returns items that are pareto-optimal.

Given the large set of preferences our implementation generates, one can easily see that PreferenceSQL will likely have to execute many sub queries until it finds Products that are pareto-optimal, leading to the observed execution times. Based on these considerations, the large speedup for the reduced number of preferences is plausible, too, although our utility-based approach also profits from the simplified queries and therefore retains a speed advantage.

In addition to the runtime issues, we also found it quite hard to control the number of query results that PreferenceSQL returns. Often, a significant portion of the entire database content is returned when all Attributes are used as preferences. Since all of the returned Products are considered to be pareto-optimal, they cannot be ordered reasonably for presentation.

9.2.1.3 Complete Dialogue Step

To give an impression of the overall system performance, we provide additional measurements for a complete "dialogue step", which we define as the time that is spent "within" our Java servlet to process the HTTP request. This includes the times for updating the Bayes network, getting recommendations, and all other overheads apart from client-server network latency. Again, the following chart shows the median values from the same experimental setup as we used for the two preceding sections. We used "Likelihood Weighting" with 8,000 samples and our utility-based approach with all Attributes considered as our "standard" configuration, and show additional variations of the number of samples and consideres Attributes.

We can see how the number of taken samples can serve as a simple parameter to control the overall system performance. Above, we have chosen the number of 8,000 samples because it consistently leads to 200–300ms execution time for the complete dialogue step. Assuming a network latency of 200ms at maximum (which is a conservative estimate if we consider a LAN or even a DSL connection), this leads to a user-perceived delay of less than half a second which we consider acceptable.

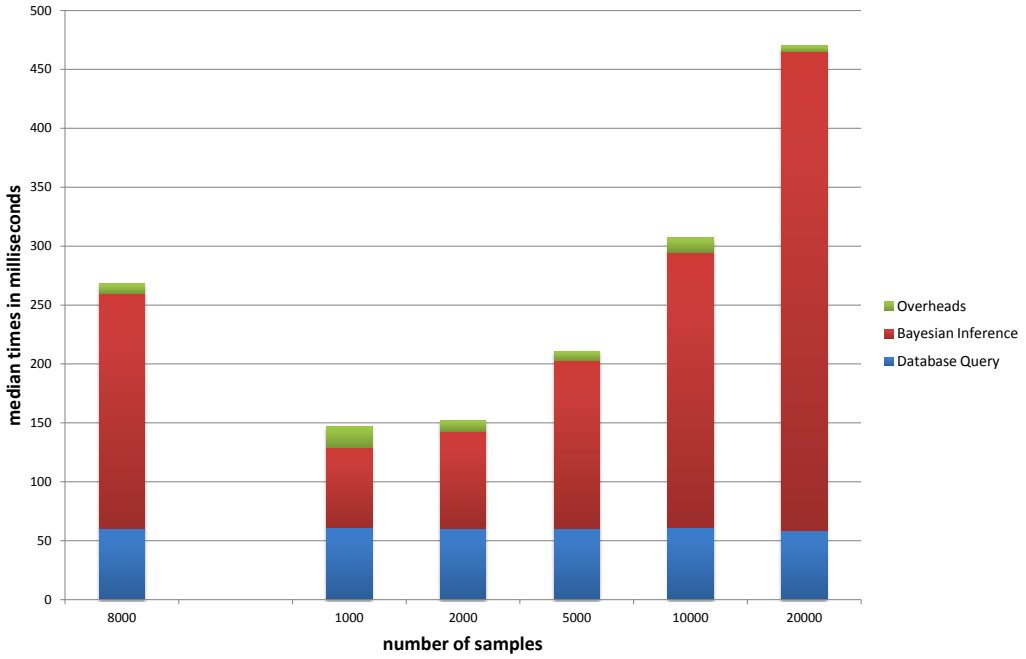

Figure 9.4: Performance measurements for complete recommendation dialogue steps, showing median values

If desired, the system might be extended to increase the number of samples for the "later" stages of the dialogue. Since the algorithms run faster if the Bayes network contains more evidence, the time that is gained in this way may be spent for increasing the inference quality, based on the consideration that the accuracy of the predictions is more important when nearing the end of the dialogue as the buying decision will be close. It is even possible to switch to the non-approximative clustering algorithm (cf. [Alt09]). However, [Alt09] also shows that the increase in quality of the probability estimations is generally not significant once, say, a few thousand samples have been executed. Our SMILE library defaults to 1,000 samples and our standard of 8,000 samples leads to already very small variances in the posteriori probability estimations.

All measurements reported above relate to the full real-life use case described in example 3.4.1. For completeness, we did also measure performance in our "movies" domain model, which is significantly smaller. Since we did not have representative customer dialogues available for this use case, the measurements were conducted on fully random dialogues. As would be expected, all parts of the system perform notably faster, leading to runtimes as shown in table 9.1.

Measurement	Median Duration	Notes
Database Query	16ms	Utility-based approach, all Attributes
Bayes Inference	59ms	Likelihood Weighting, 8,000 samples
Dialogue Step	79ms	= Database + Bayes + Overhead

Table 9.1: Measurements for the movies domain

9.2.1.4 Project Experience

Apart from our own experimental setup, we also conducted runtime measurements for the recommender system prototype built in the context of the software landscape of our industry partner. This implementation is based on the .NET framework and uses Microsoft SQL Server (via ADO.NET) as its database backend. The measurements taken in this setup can be considered to be under the most realistic circumstances that we were able to create, using production-level servers located in the datacentre of our industry partner.

The goal for our application was to achieve a "round trip", i.e., the time period from submitting the answers to a given question to displaying the next question and the corresponding recommendations including all associated overheads, of less than 1500ms. This requirement comes from the fact that about 1.5 seconds would be covered by transitioning animations on the client device and therefore can be regarded as meaning no noticeable delay for a user of the system.

Through various implementation optimizations, we were able to achieve an average "round trip" of 900ms, a significant improvement from about 4500ms that we measured for the first working version.

Of this time, 590ms are spent "inside" the actual recommender web service logic, the rest of the time is consumed by additional tasks like querying of product information (including multimedia data) for presentation, network latency, and other overheads.

As one would expect from our lab experiments, these 590ms are primarily separated into the time spent for the Bayesian inference and executing the recommendation database query. Consistent with our other measurements, the Bayesian inference takes 275ms, whereas the database queries use 260ms.

In this setup, the SQL queries were implemented as stored procedures and the main performance cost was to populate their large parameter lists. Also, the used database was a shared remote machine so that we have to assume some network latency (we

cannot influence things like pooling of database connections in ADO.NET) as well as a certain base load that other users created on the database server.

In summary, our measurements proved that the recommender system exhibits a runtime performance that is fully adequate for its intended usage in a realistic industry configuration.

9.2.2 Memory Usage

Our prototypical web application / web service generally uses a modest amount of memory, considering the fact that it is designed to run on "server" hardware. The largest amount of memory is consumed by the Bayesian inference engine, more concretely by its internal representation of the Bayes network itself.

We have measured the amount of memory used by a single instance of the Bayes network corresponding to the real-world use case (cf. example 3.4.1) to be in the order of magnitude of 100MB.

The SMILE software library requires thread-exclusive access to a Bayes network instance for reliable behaviour. Therefore, in order to support several parallel user sessions, the recommender system must keep several instances of the Bayes network in memory simultaneously, increasing the memory requirements correspondingly.

However, if we regard a "user session" as the entire recommendation process, i.e., the complete dialogue, including the times waiting for the customer to select answers etc., it becomes obvious that the Bayes network is only used a fractional amount of time during the session. Consequently, it would be an enormous waste of resources to keep one instance of the Bayesian network per user session. Instead, we opted to create a "network pool" that contains a fixed number of Bayes networks that are assigned to user sessions on-demand for the short duration of inference operations only and reset afterwards. This approach performs significantly better than, e.g., instantiating fresh Bayesian network objects for each request.

The size of the pool serves as an easily modifiable parameter to balance the consumed memory against the maximum number of allowed parallel requests to the inference engine. Note that the number of concurrent user sessions generally can be much higher since only very few sessions will require use of the inference engine at any given point in time.

Concretely, we found it a good practice to set the size of the network pool equal to the number of threads that the server can execute in parallel. At this point, the Bayesian inference becomes CPU-bound and no further gains in throughput can be expected from permitting more parallelism. In other words, given a concrete server machine with two quad-core CPUs that we had access to, the pool size was limited to 8 Bayesian networks, allowing for maximum parallelism with a measured overall memory consumption of around 1GB.

9.3 Limitations

Our evaluations also turned up some limitations of our approach, some of which can be said to be part of the design, whereas others were only found during practical use.

The dialogue is only specified indirectly, using the domain model, as the metamodel provides no means to explicitly structure the dialogue into a series of questions or similar. This means that a domain expert must ultimately concede the control of the dialogue flow to the recommender system which may not be desired in every cases. Some questions, for example, may be intuitive successors of others and the dialogue manager might fail to infer this relationship automatically. On the other hand, one might argue that the inferred dialogue sequence is in fact superior to an arbitrarily defined "logical" ordering.

Our approach relies on a relatively "rich" product model. It is doubtful how adequate the utility function would be, if the Articles were described by only very few Attributes.

The utility function allows only purely relative conclusions. It is not sensible to compare the calculated utility values from different states of the dialogue and in particular it is impossible to use the utility function for cross-domain recommendations. Such an approach would require intensive effort to merge the different domain models beforehand. Also, the numercial utility value does not allow conclusions about the absolute quality of the recommendation (i.e., it is not possible to provide a measure like the well-known "1 to 5 stars" rankings for our recommendations).

As already described in section 8.1, the degree of automation that can be achieved for the model maintenance is dependent on the quality of the available product data. In our use case, it was necessary to invest more manual work than was originally expected.

9.4 Conclusion

In summary, the evaluations show that our recommender system approach is suitable for its intended purpose. The recommendation quality is comparable to a dedicated scoring model for representative customer groups built by experts during a study from a market research institute. We argue that the market study nicely shows the real strength of our approach: Whereas the institute had to tediously build one dedicated scoring model / utility function per customer group, our system determines a personalized utility function for each single customer, even if he/she is not fully typical.

Beyond the day-to-day maintenance of the domain model by our industry partner as described in chapter 8, our student workshops revealed the simplicity and efficiency of the modelling approach. High-school students were able to understand the approach and build basic recommendation models during workshop session that lasted only a few hours. In addition, the workshops serve as principal evidence for the transferability of our approach to domains beyond mobile telecommunications.

Furthermore, our measurements show that the recommender system exhibits adequate performance to be used in the envisioned web-application scenario. Supported by additional lab measurements, benchmarks conducted in a realistic scenario at our industry partner show roundtrip times of less than one second, which are fully acceptable for interactive usage in our prototype. In particular, our measurements show that the utility-based database queries against a standard SQL database system compare favourably to PreferenceSQL and that the memory requirements for using Bayesian inference are controllable.

10 Further Work

10.1 Enhance Interaction Paradigm by Incorporating Explanations

In section 6.5, we described the use of explanations to inform the customer about some of the reasoning behind the behaviour of the recommender system. It would be interesting to expand the use of explanations beyond their plain informative character that they have in our current approach towards a more "active" use in dialogue management.

We envision that the dialogue could – after a brief sequence in the conventional fashion – continue by selecting questions based on explanation-based criteria, such as an observed discrepancy between the explanation and the predicted or actual answers of a customer. The customer could then be given the choice to "fix" these discrepancies before moving on through the regular dialogue. Furthermore, (re-)answering a question could immediately lead to feedback about how the answer has changed the recommendations and uncover new questions that now require "fixing" by the user.

10.2 Derive Maintenance Necessity Automatically

It has been observed that our usage of Bayesian networks completely excludes learning-based approaches, which is somewhat uncommon amongst the pertinent scientific community. In part, this was due to the fact that no data to learn from was available for our use case. Assuming the availablity of such information, let us consider some ways where learning techniques could be used to improve the recommender system:

- Discrepancies between the generated recommendations and the actually bought products can be uncovered. The responses to such an event may vary from a notification of the domain model maintainer to automated analysis resulting in suggestions of how to adapt the Bayesian network (ranging from changes in the

conditional probability tables to the introduction of new dependencies, i.e., new Influences in the domain model).

- Also, analysis of dialogue protocols can detect cases where the predicted answers differ significantly from the actual answers, likewise leading to manual or automatic adaptations of the domain model.

It would also be interesting to investigate techniques to generate the Bayesian network completely from history data, beginning with the conditional probability distributions and possibly even the network structure itself. We are, however, convinced that an expert-designed domain model will always have a certain influence on the Bayesian network, since the lack and possible inapplicability of history data was the very basis of our project. So, even when historic data is available, it remains doubtful that future marketing strategies could be designed without appropriate expert input. The *combination* of expert-provided and learnt probability distributions may be a promising field for further research.

10.3 Allow More Complex Influence Relationships

In section 8.4, we discussed our move from an "AND-based" to the "OR-based" approach of constructing the conditional probability tables of the Bayes network. While this has proven to be an adequate choice for the investigated use cases and is motivated by the discussed deficiencies of the AND-based calculation, it nevertheless seems somewhat arbitrary to restrict ourselves to a very specific way of constructing the probability tables.

To this end, we could imagine a more comprehensive way to specify the interrelations within a set of parent Influences for a given ReasoningElement that incorporates logical operators such as "AND", "OR", and "NOT".

In other words, assuming a set of Influences $\{I_A, I_B, I_C, I_D\}$, the current approach combines these using "OR", conceptually denoted as: $I_A \vee I_B \vee I_C \vee I_D$. The envisioned extension could allow more fine-grained specifications like, for example, $(I_A \wedge I_B) \vee (I_C \wedge I_D)$, and build a conditional probability table that reflects the specification.

It should be noted, however, that such an extension adds a significant complexity for both the domain modeller and the editor application (cf. section 8.2) which would have to be weighted against the added expressiveness that the extension provides. The significant effort to put the extension into practice, combined with the fact that

it was unclear whether a concrete use case would be found in our project, led us to not pursue the idea further in the course of the dissertation.

11 Summary

In this dissertation, we presented a novel metamodel-based approach to specify conversational recommender systems. The metamodel provides means to specify customer properties such as desires, needs, or expectations, the technical properties of the products in the domain, and the interrelationships between both. We can use domain models built upon the presented metamodel to generate all domain-dependent components of the recommender system automatically, which enables efficient maintenance.

The domain model is the foundation for a dialogue management component based on statecharts. It allows for a flexible organisation of the dialogue and is at the same time easy and intuitive to implement. In particular, the approach supports mixed-initiative dialogues, personalised dialogue paths and transparent belief revision that are necessary requirements for a conversational recommender system to be also used to assist salespersons at the point-of-sale.

Furthermore, the domain model is used to generate the system's central inference engine that relies on Bayesian networks as its underlying formalism. The inference engine provides predictions and relevance estimations for the dialogue questions and derives utility values for the technical attributes of products. The Bayesian network handles belief revision and incomplete knowledge common to the expected usage scenario. On the other hand, it can be used to explain parts of the system's behaviour such the question choosing strategy and the generated recommendations in terms that are understandable by the users.

To recommend products, the approach uses the usefulness estimations from the inference engine as inputs for a multi-attribute utility function. The function is executed on a conventional relational database to return a list of products that is ordered according to the customer's wishes. Our evaluations showed that the quality of this individualised ranking is comparable to an expert-created manual scoring model.

In addition to being implemented as a research prototype, the approach has been put into practice in an enterprise environment during a cooperation with a local industry partner to provide recommendations in the mobile telecommunications domain. An

evaluation by an external market research institute confirmed the suitability of the approach for the use case and measurements showed adequate performance in a realistic scenario. Additional evaluations demonstrated the general applicability of the approach to further business domains.

By using the implemented recommender system on a daily basis at our industry partner, we were able to demonstrate that many maintenance tasks can be solved efficiently and even semi-automatically. Additionally, a number of student workshops suggest that the developed modelling concepts are simple and efficient to use.

When comparing the approach with the works of other researchers within the recommender systems community, we find that our solution provides advantages particularly in the flexibility of the dialogue management, resulting from the fact that the dialogue is derived from the domain model rather than being specified explicitly. On the other hand, conventional collaborative or item-based recommender systems form an excessively well researched field that has proven its strengths in many practical applications. We would assume that such systems will be preferred over the approach presented here, if the surrounding conditions permit their use. In particular, such systems can handle cross-domain recommendations transparently and have the potential of even lower maintenance requirements, due to their implicit data gathering.

However, as our use case demonstrates, there are domains that cannot be adequately covered by collaborative recommender systems, because they fail to build an appropriate customer profile or rely on an unsuitable interaction model. A conversational recommender system relying on the approach presented in this thesis can be applied successfully to those scenarios.

To summarize, our work is built around the following key contributions:

- We have described a *novel architecture for conversational recommender systems* that relies on a simple, yet powerful domain modelling instead of having to use customer profiles. Systems built using this architecture can be successfully applied to problem domains that are considered to be hard to solve for other recommendation approaches.
- To this end, we have presented a *new approach to elicit customer preferences* that uses a flexible dialogue that is derived from the domain model, thereby enabling efficient maintenance.
- The described approach relies on an inference engine that *combines uncertain reasoning with database query technology* by using causal relationships specified

in the domain model. We have described our choice of a specialized variant of the "Noisy-OR" paradigm for the generation of Bayesian networks and have shown that Bayes networks using this variant are a suitable formalism for our approach, with respect to both its practical adequacy and its runtime and memory performance.

Our work combines these contributions into a system that enables product recommendations in areas that are currently unreachable for more conventional recommender technologies. Finally, it can be said that our contributions provide a suitable solution for the targeted use case and have proven their adequacy in an industrial strength environment.

A Bibliography

[AFF+02] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, M. Meyer, G. Petrone, R. Schaefer, W. Schuetz, and M. Zanker. Personalizing online configuration of products and services. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, 2002.

[AFF+03] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, G. Petrone, R. Schaefer, and M. Zanker. A Framework for the Development of personalized, distributed Web-Based Configuration Systems. *AI Magazine*, 24:93–110, 2003.

[ALS10] J.C. Anderson, J. Lehnardt, and N. Slater. *CouchDB, The Definitive Guide*. O'Reilly Media Inc., Sebastopol, 2010.

[Alt09] B. Altmann. Erweiterte Bayessche Modellierung von Domänenmodellen in einem Beratungssystem. Master's thesis, University of Passau, Chair for Information Management, 2009.

[BD04] B. Bruegge and A.H. Dutoit. *Object-Oriented Software Engineering*. Pearson Education Inc., 2004.

[BF08] M. Beck and B. Freitag. Weighted Boolean Conditions for Ranking. In *Proceedings of the ICDE Workshop on Ranking in Databases (DBRank'08)*, 2008.

[BRF07] M. Beck, S. Radde, and B. Freitag. Ranking von Produktempfehlungen mit präferenz-annotiertem SQL. In Alfons Kemper et al., editor, *Proceedings der 12. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW 2007)*, volume 103 of *Lecture Notes in Informatics*, pages 82–95, Aachen, Germany, March 2007. Gesellschaft für Informatik. in German.

[BRJ05] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, 2005.

[Bur00] R. Burke. Knowledge-based Recommender Systems. *Encyclopedia of Library and Information Science*, 69(32):180–200, 2000.

[Bur07] R. Burke. Hybrid Web Recommender Systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *LNCS*, pages 377–408. Springer Verlag, 2007.

[CD00] J. Cheng and M.J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. In *Journal of Artificial Intelligence Research (JAIR)*, volume 13, pages 155–188. Elsevier Science Publishers B.V. (North Holland), 2000.

[Cen09] Centrum für Marktforschung der Universität Passau. Nutzerprofile für Mobiltelefone und Software-Evaluation: Zwischenbericht. October 2009.

[Cen10] Centrum für Marktforschung der Universität Passau. Nutzerprofile für Mobiltelefone und Software-Evaluation: Ergebnisbericht. January 2010.

[CGL01] D. Calvanese, G. De Giacomo, and M. Lenzerini. Ontology of Integration and Integration of Ontologies. In C.A. Goble, D.L. McGuinness, R. Möller, and P.F. Patel-Schneider, editors, *Description Logics*, volume 49 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[CL07] Y. Cao and Y. Li. An intelligent fuzzy-based recommendation system for consumer electronic products. *Expert Systems with Applications*, 33:230–240, 2007.

[CP05] L. Chen and P. Pu. Trust Building in Recommender Agents. In *Proceedings of the 1st International Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces (WPRSIUI-05)*, pages 135–145, Reading, UK, 2005.

[dCFLH04] L.M. de Campos, J.M. Fernandez-Luna, and J.F. Huete, editors. *Information Processing and Management: Special Issue on Bayesian Networks and Information Retrieval*, volume 40. Elsevier, 2004.

[DG93] P. Dagum and A. Galper. Additive Belief-Network Models. In *Proc. of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI1993)*, 1993.

[DG04] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters . In *Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, December 2004.

[DMIZ97] F. J. Díez, J. Mira, E. Iturralde, and S. Zubillaga. DIAVAL, a Bayesian expert system for echocardiography. *Artificial Intelligence in Medicine*, 10:59–73, 1997.

[DP96] P. Domingos and M.J. Pazzani. Beyond independence: conditions for the optimality of the simple Bayesian classifier. In *Proceedings of the 13th International Conference on Machine Learning (ICML 96)*, Bari, Italy, 1996.

[Düm03] L. Dümbgen. *Stochastik für Informatiker*. Springer Verlag, 2003.

[EFHB10] S. Edlich, A. Friedland, J. Hampe, and B. Brauer. *NoSQL, Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Carl Hanser Verlag München, 2010.

[Erl06] T. Erl. *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall, 2006.

[ES93] L. M. Ellram and S. P. Siferd. Purchasing: The cornerstone of the total cost of ownership concept. *Journal of Business Logistics*, 14(1), 1993.

[FC90] R. Fung and K.-C. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence*, volume 5, pages 209–219. Elsevier Science Publishers B.V. (North Holland), 1990.

[FdF94] R. Fung and B. del Favero. Backward simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence*, Tenth Conference, pages 227–234. Morgan Kaufmann, San Francisco, CA, 1994.

[FFG+07] A. Felfernig, G. Friedrich, B. Gula, M. Hitz, T. Kruggel, R. Melcher, D. Riepan, S. Strauss, E. Teppan, and O. Vitouch. Persuasive recommendation: Exploring Serial Position Effects in Knowledge-based Recommender Systems. In *Proceedings of the 2nd Internation Conference of Persuasive Technology*, volume 4744, Stanford, USA, 2007. Springer Verlag.

[FFJZ06] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An Integrated environment for the Development of Knowledge-Based Recommender Applications. *International Journal of Electronic Commerce*, 11(2):11–34, 2006.

[FLNP00] N. Friedman, M. Linial, I. Nachman, and D. Peer. Using Bayesian Net-

works to Analyze Expression Data. *Journal of Computational Biology*, 7(3/4):601–620, 2000.

[Fra03] David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press, 2003.

[Fre10] Burkhard Freitag. Präferenzen und Ranking in Informationssystemen. Lecture script of the Chair for Information Management, University of Passau, 2010.

[Fri97] J.H. Friedmann. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[GBH09] B. Gipp, J. Beel, and C. Hentschel. Scienstein: A Research Paper Recommender System. In *Proceedings of the International Conference on Emerging Trends in Computing (ICETiC09)*. Kamaraj College of Engineering and Technology India, IEEE, 2009.

[Hel11] M. Helm. Semantic Search: Semantische Suche auf taxonomisch strukturierten Daten im verteilten Archivsystem MonArch. Master's thesis, University of Passau, Chair for Information Management, 2011.

[Hen88] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence*, volume 2, pages 149–163. Elsevier Science Publishers B.V. (North Holland), 1988.

[HHHC04] S. M. Hsieh, S. J. Huang, C. C. Hsu, and H. C. Chang. Personal document recommendation system based on data mining techniques. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '04, pages 51–57, Washington, DC, USA, 2004. IEEE Computer Society.

[HKTR04] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[HP09] R. Hu and P. Pu. A Comparative User Study on Rating vs. Personality Quiz based Preference Elicitation Methods. In *Proceedings of the International Conference on Intelligent User Interfaces*. ACM, 2009.

[HP10] R. Hu and P. Pu. A study of Building Profiles for Different Objects using Personality Quizzes in Recommender Systems. In *Proceedings of the 18th International Conference on User Modeling, Adaptation and Personaliza-*

tion (UMAP), volume 6075 of *LNCS*. Springer Verlag, 2010.

[Höl11] G. Hölbling. *Personalized Means of Interacting with Multimedia Content*. PhD thesis, University of Passau, 2011.

[JSLZ03] J.Z. Ji, Z.Q. Sha, C.N. Liu, and N. Zhong. Online recommendation based on customer shopping model in e-commerce. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI)*, 2003.

[JSSW95] A. Jameson, R. Schaefer, J. Simons, and T. Weis. Adaptive Provision of Evaluation-Oriented Information: Tasks and Techniques. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[JZFF11] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2011.

[Kar11] J. Karl. Nutzung einer NoSQL-Datenbank für Produktempfehlungen in einem elektronischen Beratungssystem. Bachelor's thesis, University of Passau, Chair for Information Management, 2011.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 311–322. Morgan Kaufmann, 2002.

[Kie05] W. Kießling. Preference Queries with SV-Semantics. In *Proceedings of the 11th International Conference on Management of Data (COMMAD)*, pages 15–26. Computer Society of India, 2005.

[KK02] W. Kießling and G. Köstler. Preference SQL – Design, Implementation, Experiences. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, 2002.

[Köl99] A. Kölzer. Universal Dialogue Specification for Conversational Systems. In J. Alexanderrson, L. Ahrenberg, K. Jokinen, and A. Joensson, editors, *Special Issue on Intelligent Dialogue Systems*, number 9 in News Journal on Intelligent User Interfaces. ETAI, 1999.

[Kor09] Y. Koren. The BellKor Solution to the Netflix Grand Prize. Technical report, Netflix Prize Documentation, 2009.

[KS03] Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: The State Of The Art. *Knowledge Engineering Review*, 18(1):1–31, 2003.

[Kul09] H. Kulovits. Recommender Systems in Preservation Planning. In *Proceedings of the 9th Workshop on Data Analysis (WDA-09)*, Certovika, Slovakia, July 2009.

[LB06] J. Leite and M. Babini. Dynamic Knowledge Based User Modeling for Recommender Systems. In *Proceedings of the ECAI-06 Workshop on Recommender Systems*, 2006.

[Lik32] R. Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 22(140):55ff, 1932.

[LV07] T.D. Loboda and M. Voortman. About GeNIe & SMILE. Available online at `http://genie.sis.pitt.edu/about.html`, 2007. Last accessed on 2012-03-27.

[MN98] A. McCallum and K. Nigam. A Comparison of Event Models for Naive Bayes Text Classification. In *Proceedings of the AAAI Workshop on Learning for Text Categorization*, Madison, USA, 1998.

[MR07] T. Mahmood and F. Ricci. Learning and adaptivity in interactive recommender systems. In *Proceedings of the 9th International Conference on Electronic Commerce*. ACM, 2007.

[Pea97] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1997.

[PHC07] M.-H. Park, J.-H. Hong, and S.-B. Cho. Location-Based Recommendation System Using Bayesian User's Preference Model in Mobile Devices. In *Proceedings of the 4th International Conference on Ubiquitous Intelligence and Computing (UIC)*, 2007.

[PPNH94] M. Pradhan, G. Provan, B. Niddleton, and M. Henrion. Knowledge Engineering for Large Belief Networks. In *Proceedings of the Conference on Uncertainty in AI*, 1994.

[RBF07] S. Radde, M. Beck, and B. Freitag. Generating Recommendation Dialogues from Product Models. In *Proceedings of the AAAI Joint Workshop on Intelligent Techniques for Web Personalization and Recommender Systems in E-Commerce*, pages 126–129, Vancouver, Canada, July 2007. AAAI Press.

[RD06] F. Ricci and J. Delgado, editors. *Special issue on Travel Recommender Systems*, volume 6 of *Information Technology and Tourism*, 2006.

[RF07] S. Radde and B. Freitag. SIPREACT - Kontextsensitive Beratungssysteme. In R. Koschke, O. Herzog, K.-H. Rödiger, and M. Ronthaler, editors, *Proceedings des GI-Workshops zu Situierung, Individualisierung und Personalisierung*, volume P-109 of *GI-Edition LNI*, pages 259–262, Bremen, Germany, September 2007. Gesellschaft für Informatik. in German.

[RF10] S. Radde and B. Freitag. Using Bayesian Networks To Infer Product Rankings From User Needs. In *Proceedings of the UMAP Workshop on Intelligent Techniques for Web Personalization and Recommender Systems (ITWP'10)*, Big Island, Hawaii, June 2010.

[Ric02] F. Ricci. Travel recommender systems. *IEEE Intelligent Systems*, 2002.

[RKF08] S. Radde, A. Kaiser, and B. Freitag. A Model-Based Customer Inference Engine. In *Proceedings of the ECAI Workshop on Recommender Systems*, Patras, Greece, July 2008. ECCAI.

[RN10] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Inc., 2010.

[RRSK11] F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor, editors. *Recommender Systems Handbook*. Springer Verlag, Berlin, 2011.

[RZF09] S. Radde, B. Zach, and B. Freitag. Designing a Metamodel-Based Recommender System. In T. Di Noia and F. Buccafurri, editors, *Proceedings of the 10th International Conference on E-Commerce and Web Technologies (EC-Web 2009)*, volume 5692 of *LNCS*, pages 264–275, Linz, Austria, September 2009. Springer Verlag.

[SB01] S. Schmitt and R. Bergmann. A Formal Approach to Dialogs with Online Customers. In *Proceedings of the 14th Bled Electronics Commerce Conference*, pages 309–328, 2001.

[SB11] O.C. Santos and J.G. Boticario, editors. *Educational Recommender Systems and Technologies: Practices and Challenges*. IGI Global, 2011.

[SK09] X. Su and T.M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence (AAI)*, 2009, 2009.

[SP90] R.D. Shachter and M.A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence*, volume 5, pages 221–231. Elsevier Science Publishers B.V. (North Holland), 1990.

[Sta10] Statistisches Bundesamt. Altersstruktur der Bevölkerung (Stand: 31.12.2009) im Vergleich zum Vorjahr. Available online at `http://de.statista.com/statistik/daten/studie/1351/umfrage/altersstruktur-der-bevoelkerung-deutschlands/`, Nov 2010. Last accessed on 27.09.2011.

[Tay10] M.M. Taye. State-of-the-Art: Ontology Matching Techniques and Ontology Mapping Systems. *International Journal of ACM Jordan*, (3):8, 2010.

[WE86] D. Winterfeldt and W. Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, 1986.

[Wie03] R.J. Wieringa. *Design Methods for Reactive Systems*. Morgan Kaufmann, 2003.

[YD03] C. Yuan and M.J. Druzdzel. An Importance Sampling Algorithm Based on Evidence Pre-propagation. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 624–631. Morgan Kaufmann: San Mateo, CA, 2003.

B List of Figures