Specification of Adaptive Client-Tailored Product Models

Tibor Bosse, Fiemke Both, Mark Hoogendoorn, and Jan Treur
Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{tbosse, fboth, mhoogen, treur}@cs.vu.nl

Abstract

Traditionally, product models often have a rigid nature, both with respect to the manner in which they are initially tailored to clients, and to the way they are maintained over time. Especially when such products are offered at a highly interactive medium such as Internet in the form of web services, addressing such aspects might be a necessity to obtain a competitive advantage. To cope with these rigidity problems, this paper proposes a dynamic approach to product models, which supports an ongoing interaction process that continuously adapts a product to the background and desires of the client. The generic approach has been formalised and tested in a case study in the domain of car insurances.

1. Introduction

Nowadays, more and more products are being offered on the web in the form of web services. Financial institutions for example, also intend to offer their products in such a form in the near future. Within the area of web services, a lot of research has been conducted to compose such web services [12, 13, 14]. In order to maximise the probability that the product offered as a web service is indeed selected as part of such a composition, the company should offer the best product configuration it can. Such an offer can only be made if the product is fully tailored towards the desires and characteristics of the customer. Hence, a flexible and adaptive approach to obtain and maintain a product configuration is needed to maximise the success of offering a product as a web service.

Usually a product model specifies a product by a number of attributes and values for them. Financial products for example, are often described by specifications of values for certain attributes, for example for an insurance the coverage, premium, and way of paying. However, as financial products usually extend over time, this temporal dimension has to be covered as well in a product model. For example, for an insurance, the attribute 'no claim reduction percentage' is not a static product aspect, but changes over time, depending on dynamic interactions of the client and its environment. Therefore the specification of such an aspect can not be based (only) on specification of a value for a certain attribute. Instead of such a simple specification, a more complex temporal relationship has to be specified taking into account events and attributes at different points in time. For example, the no claim reduction percentage at a certain point in time depends on the claims made at earlier time points in the past, which depends on the personal situation of the client over time. Also for other products such temporal patterns play an important role.

The dynamic aspects of product models have not always been given the central position as would be realistic and useful. Instead, product models often have been forced in a static format entailing rigidity of two types: (1) rigidity in the manner most products are initially fit to clients and (2) rigidity in the way they are maintained over time after a client has accepted them. More specifically, examples of rigidity of type (1) are found in the way how product alternatives are offered as specific predefined packages or combinations of basic product attributes, for example, specific types of saving accounts. The client has to make a choice between a course-grained couple of predefined alternatives, and is not allowed to propose a different alternative that may be more tailored to (the development of) his or her own situation. Examples of rigidity of type (2) are found in situations that the development of the client's situation makes the choice for a certain product as made in the past no longer rational. These developments may relate to various aspects of personal life varying from earning money by a well-developing career to marriages, divorces, birth, etc. A simple example is a situation in which the amount of savings increases to such an extent that a different type of saving would be more economical for the client. Often such a client is made aware of this by a competitive financial organisation, instead of by the own financial organisation.

As a solution to the situation sketched above, this paper makes a first attempt to make product models more

dynamic and tailored to specific clients. This dynamic perspective is explained in detail in Section 2. Based on this perspective, Section 3 introduces a global outline of our approach to (re)design product models in a dynamic manner. Next, Section 4 introduces the formal modelling concepts underlying the approach. Using these concepts, the approach itself is formalised in Section 5. Section 6 addresses a case study, in which the approach is applied in the domain of car insurances. Section 7 concludes the paper with a discussion.

2. Putting Dynamics into Product Models

To fully cope with the dynamic aspects products have and to avoid the types of rigidity as discussed above, a different perspective on product models is needed. From this dynamical perspective, a product model is basically a specification of an adaptive dynamical system, which continuously tailors itself to the client's situation and the wider environment, and developments therein. Then product aspects as offered to the clients are not primarily static basic attributes such as interest rates or insurance premiums, but temporal rules that specify how such basic attributes will behave over time, in interaction with developments in the client's life and the wider environment.

To obtain such a dynamical perspective in an operational form, the following desiderata can be formulated:

(a) Fine-grained client-sensitive design of products
(b) Continuous adaptation of products to developments in the client's situation and the environment

Concerning (a), it should be possible, at any point in time, to tailor a product in a fine-grained manner to the client's situation and developments therein. A course-grained approach of choosing between a small number of predefined alternatives will not be sufficient. In contrast, it is required that a product can be designed in a more precise manner based on the specific requirements imposed by developments in the specific client's situation, for example by tuning certain numerical parameters of the product with a certain precision.

Concerning (b), the fine-grained and client-sensitive design process should be repeated regularly over time to be able to continuously adapt the product to developments in the client's situation and the wider environment. Thus instead of a design process at one point in time, a repeated redesign process has to be specified within the product model indicating the ongoing adaptation of the product to the client's life.

Desiderata (a) and (b) can be realised when a dynamic product model is specified as an adaptive dynamical system that takes the form of a continuous product (re)design process model.

The area of adaptive dynamical systems is often addressed by mathematical techniques based on differential equations. Such a purely numerical perspective has serious shortcomings with respect to expressivity, in particular for qualitative aspects and complex temporal relationships between them. In the next sections it is shown how an integrated modelling approach can be used to specify a product model as an adaptive dynamical system.

3. Specifying Continuous Product Design

To specify a product (re)design process model from the perspective indicated above, a number of elements are relevant, concerning the processes, ontologies and types of knowledge. A dynamic product model format will be defined that incorporates these elements.

First, three main subprocesses can be distinguished:

- design requirements determination (DRD)
- design object determination (DOD)
- design process coordination (DPC)

These subprocesses are inspired by the Global Design Model (GDM), introduced in [5], and worked out in a simulation model in [4]. Here, the design requirements determination process first identifies requirements based on characteristics and preferences of the client and his or her situation. Next, these requirements are refined into more specific requirements relating more directly to certain basic attributes of a design object (a product). The process to generate a design object description determines values for basic attributes of a suitable product based on the specific requirements as identified. Moreover, from these basic attributes, values for dependent attributes are derived. The design process coordination process provides strategic input for the design process to be performed.

Second, to represent the relevant information and knowledge, different types of ontologies are needed:

- ontology to describe the environment
- ontology for client characteristics and preferences
- ontology for temporal relationships
- design requirement ontology
- design object ontology
- design process coordination ontology

The ontology to describe the environment and its development includes elements referring, for example, to different aspects of the economical development and the financial market. The ontology for client characteristics and preferences is the basis for client models. The ontology for temporal relationships is used on top of the other ontologies, to express dynamics. The other three ontologies are used in the three main processes within the redesign process.

Third, to model the processes, specification of different types of dynamical relations is needed:

(1) to identify requirements and to refine them into more specific requirements
(2) to relate specific requirements to basic design object attributes
(3) to determine derivable design object attributes from basic design object attributes
(4) to evaluate a design object with respect to requirements

Dynamical relations of type (1) to identify and refine requirements may involve heuristic parameters to relate one requirement to a number of more specific requirements relating to different aspects of a product. For example, if the payment per month for a mortgage is based on a part that covers the interest and a part that covers the pay-off, to get a modest payment per month, both of these parts should be kept modest, according to a certain proportion (ratio of distribution). In this refinement knowledge, also sources in the area of multi-attribute brokering and negotiation can be exploited.

Dynamical relations relating specific requirements to basic attributes of the design object (2) may have the form, for example, that based on a requirement expressing the limit values of an attribute, one of these limit values is chosen as the value of the attribute.

Dynamical relations to determine derivable design object attributes from basic design object attributes (3) may involve specific calculations of values for dependent attributes from the values of the basic attributes, for example, an insurance premium from other attributes.

Dynamical relations to evaluate a design object description with respect to requirements (4) involves, for example, how based on product attributes it can be determined whether or not a given requirement is satisfied by the design object. Moreover, it may also involve relationships to determine in how far the design object fulfils the client's characteristics and preferences as specified in the client model.

Note that the dynamical relations of type (1) are used within the DRD subprocess introduced above. Moreover, relations (2), (3) and (4) are used within the DOD subprocess.

4. Dynamic Modelling Approach

To model and analyse product dynamics, a formalisation is needed of such dynamics. Such a formalisation is introduced in this section. This formalisation uses the notions of *state* and *trace*.

States and State Ontologies State ontologies used are ClientOnt (for clients), EnvOnt (for environment), ReqOnt (for requirements) , and DODOnt (for design object descriptions). The definitions of these ontologies are shown in Appendix A[1]. The set of *ground state atoms* over an ontology Ont is denoted by GSTATOMS(Ont). A *state* S over a state ontology Ont is a mapping assigning truth values to the ground atoms $S : GSTATOMS(Ont) \rightarrow \{$ true, false, undefined $\}$. The set of all possible states over Ont is denoted by STATES(Ont).

For example, STATES(ClientOnt) denotes the set of possible states of a client, STATES(EnvOnt) the set of possible states of the environment, STATES(ReqOnt) the set of possible states of the requirements, and STATES(DODOnt) denotes the set of possible states of the product attributes (design object). A client state represents a client model at a specific point in time. An example of an aspect in a client state is the client's risk avoidance profile expressed by a risk versus expected gain proportion. An environment state represents the state of the environment at a certain time point; aspects are, for example, current values of indicators of the economy or the financial market. A requirements state represents the state of the requirements imposed by the client and environment states at a certain time point. An object state represents the state of a product at a certain time point. An example of an aspect in an object state is a certain product parameter such as the monthly amount to pay (e.g., in case of a mortgage).

Traces Traces are time-indexed sequences of states. To describe such sequences a fixed *time frame* T is assumed which is linearly ordered (e.g., the real or natural numbers). A *trace* γ over a design state ontology Ont and time frame T is a mapping $\gamma: T \rightarrow STATES(Ont)$, i.e., a sequence of states γ_t $(t \in T)$ in STATES(Ont). The set of all traces over state ontology Ont is denoted by TRACES(Ont). Depending on the application, the time frame T may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering.

For example, TRACES(ClientOnt) denotes the set of possible traces for a client, TRACES(EnvOnt) the set of possible traces for the environment, TRACES(ReqOnt) the set of possible traces for the requirements, and TRACES(DODOnt) denotes the set of possible traces for the product attributes (design object). A trace for a client represents the evolution of the client model over time. An example of an aspect in a trace for a client is the change over time of the client's risk avoidance profile, or a progressive trend in income over time. A trace for the environment represents the evolution of the environment state over time. An example of an aspect in such a trace is the trend in some indicator of the economy or financial market (for example, an increasing interest rate). A trace for an object represents the evolution of the object state over time. An example of an aspect in a trace for an

[1] http://www.cs.vu.nl/~tbosse/prodmodels/

object is the change over time of the monthly amount to be paid.

Dynamical relations The four different types of states interact with each other over time. For example, the client state and environment state both affect the requirements state. Moreover, the requirement state affects the object state, and the object state affects the future client states, which makes it a cyclic process. These temporal relations can be depicted in graphical format as shown in Figure 1. Here, the numbers that are attached to the arrows correspond to the four types of dynamical relations introduced in Section 3. As depicted, the environment state is not affected by any of the other states. However, in principle it is possible that the development of new products over time also affects the environment state via the client states (if a large scale use of these new products occurs), which would show another cycle in the dynamical system. For the time being this possibility is ignored here.

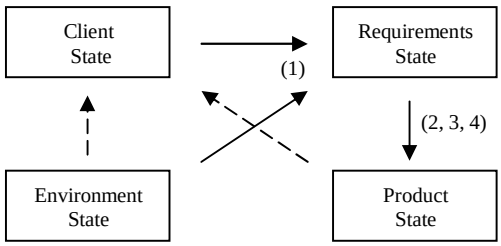

Figure 1. Global Dynamic relations

The traces of these states representing development over time can be depicted along a time axis as shown in Figure 2. Notice that here the arrows pointing downwards are specified as part of a dynamic product model. The dashed arrows pointing towards the client trace are not specified within the dynamic product model, but indicate the effect of the product and environment state on the client's state (taking place within the client).

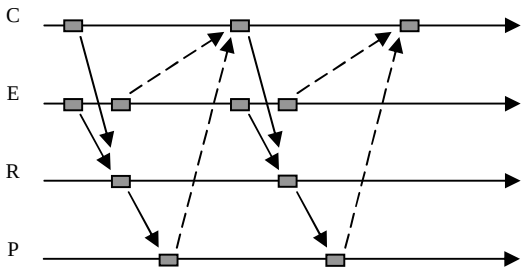

Figure 2. Traces for client C, environment E, requirements R and product P over time

The following is an example of a story as depicted here. A new client comes in, his profile is identified (client model), based on that requirements are formulated fitting this client model and a mortgage is designed with attributes tailored to his situation. At a next point in time, the client's financial situation changes due to a divorce. An updated client model is obtained, based on which new requirements are formulated and the basic attributes of the mortgage adapted to satisfy these new requirements.

Specification of Dynamic Relations To formally specify dynamic properties that express characteristics of processes from a temporal perspective, an expressive language is needed. To this end, the Temporal Trace Language TTL [3] can be used as a tool. This language can be classified as a predicate-logic-based reified temporal language; see [8, 9]. Within TTL, complex dynamic properties of design processes can be expressed. An example is the following property: "If between time point t1 and t2, for each existing requirement a product is found that fulfils it, then at time point t2 the profit will be at least x".

However, this paper only considers properties that address snapshots of the design process at a single time point, such as the property "for each client a suitable insurance is derived". Since these properties abstract away from the temporal aspect, for the moment standard first-order predicate logic can be used instead of TTL. Thus, an order-sorted predicate logic ontology Ont to describe state properties is assumed, consisting of sorts, subsort relations, constants in sorts, and functions and relations over sorts; e.g., [10]. Moreover, the usual first-order logical connectives such as $\neg$, &, $\vee$, $\Rightarrow$ $\forall$, $\exists$ are used. In future work, TTL will be used to express the more complex dynamic properties as shown above.

Executable Dynamic Properties To be able to perform automated experiments with design processes, a simple temporal logical language to specify simulation models is used. This language LEADSTO [2] enables to model direct temporal dependencies between two state properties in successive states, as occur in specifications of a simulation model (for example, if in the current state, state property p holds, then in the next state, state property q holds). This language is executable and therefore enables the automatic generation of simulated traces; for other executable temporal languages based on modal logic, see [1]. This section briefly introduces the logical format used for these LEADSTO simulation models. This executable format is defined as follows. Let α and β be state properties of the form 'conjunction of ground atoms or negations of ground atoms'. In the leads to language the notation $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, means:

If state property α holds for a certain time interval with duration g, then after some delay (between e and f) state property β will hold for a certain time interval of length h.

5. Dynamic Relations for Product Redesign

Based on the formalisation presented above, this section presents the specification of the dynamical model needed to come to a suitable design for a product based upon the input received (from the environment, or based upon the client model). First of all, it is shown how to establish requirements and to refine them into more specific requirements (i.e., dynamical relations (1) in Section 3). Thereafter, it is shown how those specific requirements relate to basic design object attributes (2). How to derive design object attributes from basic design objects (3) is presented thereafter, and finally, it is shown how to evaluate a design object with respect to the requirements (4). All specifications are in LEADSTO format.

5.1. Requirements Identification and Refinement

Within this process, it is assumed that different requirements for a product can be related to each other by means of a refinement tree. This assumption corresponds to the idea presented in [7] that (goal-oriented) requirements may be refined into more specific requirements on parts of a design object. An abstract example of such a refinement tree is depicted in Figure 3.

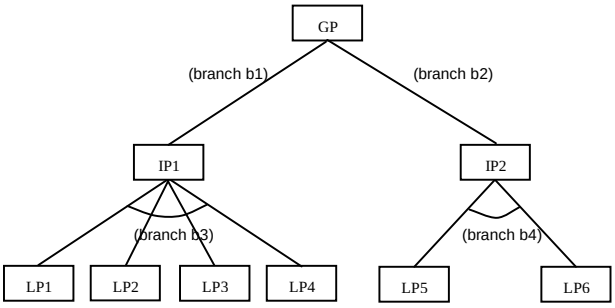

Figure 3. Example requirements tree

In this figure, requirements are represented by boxes, and refinement relations are depicted by lines between them. Different lines connected by an arc denote an AND-branching, meaning that the requirement is refined into the combination (conjunction) of requirements below. Lines without an arc denote OR-branches, indicating alternative refinements. For example, in Figure 3, GP is the highest requirement for the product. This requirement can be fulfilled by fulfilling either requirement IP1 or IP2. Furthermore, IP1 is fulfilled by fulfilling all of LP1-LP4, and IP2 is fulfilled by fulfilling both LP5 and LP6. Such refinement relations are represented by predicates such as is_a_subrequirement_of_via(IP1,GP,b1), which are part of ReqOnt (see Section 4).

The idea is that, within a design process, the top level requirements are context-dependent, and are identified by the product selling company in interaction with the client on the basis of the client's characteristics and financial indicators. Next, lower level requirements are derived, based on the following specifications:

LTP1 Requirement Refinement
This LEADSTO property (LTP) expresses that, if a requirement exists that can be refined to a sub-requirement, then this should be done by refining via the best branch.

∀p,q:REQUIREMENT, ∀b:BRANCH
is_a_current_requirement(p) ∧ is_a_subrequirement_of_via(q,p,b) ∧ best_branch_for(b,p) → is_a_current_requirement(q) ∧ requirement_refined(p) ∧ requirement_refined_via(p,b)

Here best_branch_for(b,p) can be defined in various manners. An example is to calculate the predicted production costs for all branches that refine requirement p and to select branch b for which these costs are lowest.

To formally specify the exact content of a requirement, the predicate has_expression is used. An example instance of this predicate, expressed in the TTL language, is as follows:

has_expression(LP1,
∃X:real variable_has_value(premium, X) ∧ X < 100)

This means that requirement LP1 expresses that there should exist a real value for the variable premium such that this value is lower than 100.

5.2. Relating Requirements to Design Objects

As soon as the basic requirements (i.e., the leaves of a tree such as depicted in Figure 3) have been established, Design Object Descriptions (DODs) can be found. To this end, the following specifications are used:

LTP2 DOD Generation
This property expresses that each local requirement should be satisfied by adding the best product characteristic for that requirement to the current DOD.

∀l:LOCAL_REQUIREMENT, ∀c: VARIABLE, ∀x:INTEGER, ∀v:VALUE
is_a_current_requirement(l) ∧ best_characteristic_for(c,v,l) ∧ DOD_counter(x) → current_DOD(dod(x)) ∧ part_of_DOD(c,dod(x)) ∧ variable_has_value(c,v)

Here best_characteristic_for(c,v,l) is defined, for example, by picking a value v for variable c that satisfies the inequality expressed in the requirement l. Note that there can also be multiple best values for a particular variable, for more details on the selection process between these values, see [4].

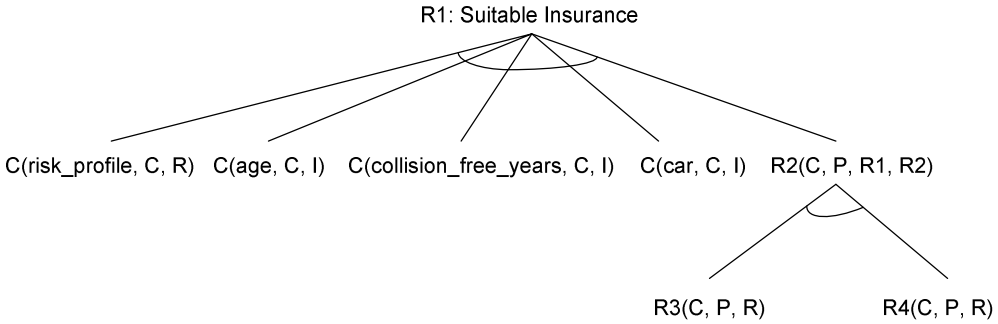

Figure 4. Requirements tree for car insurances

5.3. From Basic Design Object Attributes to Derivable Design Object Attributes

Based upon certain basic design objects attributes that have been generated using the requirements, other attributes can be derived with the dependencies that exist between these attributes. An example rule for such derivation is specified below (see Appendix A for more examples). The example expresses an explicit dependency whereby it is explicitly stated what value a variable has based upon the value of other variables. For example, for a car insurance, 4 years without collision means a no claim discount percentage of 20%.

LTP3 Dependency of type *explicit*
If a variable c1 has a dependency relation of type explicit, stating that is will get value v1 if another variable c2 has value v2, and this is indeed the case, then c1 gets value v1.

∀l:LOCAL_REQUIREMENT, ∀c1: VARIABLE, ∀c2:VARIABLE, ∀x:INTEGER, ∀v1,v2:VALUE
is_a_current_requirement(l) ∧ holds_for(l, c1) ∧ DOD_counter(x) ∧ is_dependent_on(c1, [c2]) ∧ dependency_relation_of_type(c1, [c2], explicit) ∧ dependency_between(c1, v1, [(c2, v2)]) ∧ variable_has_value(c2, v2) → current_DOD(dod(x)) ∧ part_of_DOD(c1,dod(x)) ∧ variable_has_value(c1, v1)

5.4. Evaluation of a Design Object

Finally, the generated design object can be evaluated once more, to check whether it indeed satisfied all requirements. The following is used for this:

LTP4 Requirement Satisfaction Determination
This property determines when a certain (local) requirement is satisfied by a DOD. This is the case when the current DOD contains a variable with a value that satisfies this requirement.

∀d:DOD, ∀c:VARIABLE, ∀l:LOCAL_REQUIREMENT, ∀v:VALUE
current_DOD(d) ∧ part_of_DOD(c,d) ∧ holds_for(l,c) ∧ is_a_current_requirement(l) ∧ variable_has_value(c,v) → local_requirement_satisfied(l)

6. Case Study

This section presents a case study where the approach presented above is applied to the domain of car insurances. As described earlier, the refinement relations between requirements are represented as an AND/OR-tree. For the domain of car insurances, an example of such a tree is shown in Figure 4. At the highest level in the hierarchy, requirement R1 expresses that "for each client a suitable insurance should be derived". This property is the conjunction of a number of environmental (client dependent) conditions (below called *client characteristics*), and the requirement that a suitable premium and own risk should be found (requirement R2). The latter requirement is refined by means of requiring a suitable own risk as well as a suitable premium to be chosen.

Below, these sub-requirements are worked out in more detail. Section 6.1 addresses the client characteristics, and Section 6.2 addresses requirement R2. The product (i.e. design object) is addressed in Section 6.3. After that, Section 6.4 presents an example simulation trace for the case study.

6.1. Client Characteristics

The first client characteristic, C(risk_profile, C, R), is a risk function that determines the tradeoff a particular client makes between the premium and the own risk he or she is willing to accept. In Figure 5 the dashed line indicates such a risk curve. The line indicates the preference of a person in taking a particular risk. Some people might be interested in taking a lot more risk despite the fact that the premium is not even that much lower, whereas other people might be very unwilling to take a lot more risk, even though they will get a lower premium. Using the formal ontology for variables (see Appendix A), the property can be expressed as follows:

C(risk_profile, C, R)
variable_has_value(client_characteristic(C, risk_profile), R)

This expresses that the variable regarding the risk profile of a client has a particular value, which represents the desired risk/premium dependency.

The other three client characteristics can be specified in a similar fashion, as shown below.

C(age, C, I)
variable_has_value(client_characteristic(C, age), I)

C(collision_free_years, C, I)
variable_has_value(client_characteristic(C, collision_free_years), I)

C(car, C, I)
variable_has_value(client_characteristic(C, type_of_car, I)

As can be seen, three other characteristics are specified, namely the age of the client, the number of collision free years, and finally, the type of car the client is driving expressed in horsepower (HP).

6.2. Suitable Risk-Premium Combination

Requirement R2 specifies that, given the client, an optimal combination for the premium and own risk should be chosen.

R2(C, P, R1, R2): Product on Risk/Premium Curve fitting Client risk profile
∃R1':real, R2':real, R:real
variable_has_value(product_characteristic(P, own_risk), R1) ∧
variable_has_value(product_characteristic(P, premium), R2) ∧
variable_has_value(client_characteristic(C, risk_profile), R) ∧
optimal_own_risk_premium_for_risk_profile(R1', R2', R)
∧ R1≤ R1' ∧ R2 ≤ R2'

The definition of
optimal_own_risk_premium_for_risk_profile(R1', R2', R)
is part of the background knowledge. It can be determined by the risk/premium curve, given the client's characteristics (for an example thereof, see the solid line in Figure 5) and risk profile R. The precise values can be found by deriving the point in the risk premium curve of which the tangent has the same coefficient as the risk profile of the client (the dotted line). The formula for the risk/premium curve is as follows:

$$ownrisk(premium) = \frac{50000}{premium - d} - k \qquad (1)$$

When the parameters *d* and *k* are altered, the curve shifts horizontally and vertically, respectively. The basic values of *d* and *k* are 0 and 500. The age of the client influences *k*, the number of collision free years influences *d*, and the type of car influences both parameters. A more detailed description of the calculations is given in Appendix A.

Next, requirement R2 can be refined (by an AND-branch) to requirements R3 and R4. For this example, their expressions are:

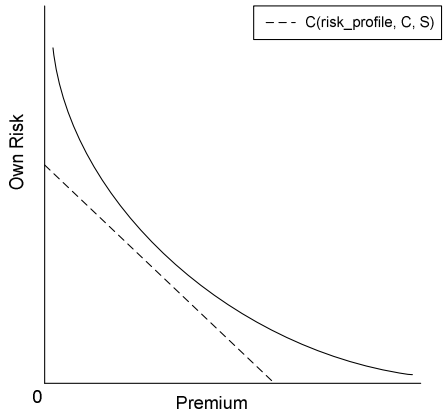

Figure 5. Example client risk profile and company risk/premium curve

R3(C, P, R): Suitable Own Risk ≤ R
∃R':real
variable_has_value(product_characterstics(P, own_risk), R') ∧
R' ≤ R

R4(C, P, R): Suitable Premium ≤ R
∃R':real
variable_has_value(product_characterstics(P, premium), R') ∧
R' ≤ R

The idea of a refinement hierarchy is that the refinement set together entails the original requirement.

6.3. Design Object

For this case study, it is assumed that a complete insurance product (or design object) is specified to consist of three parts, namely a premium value, the own risk value, and the terms type. These are specified as follows.

D(premium, P, R)
variable_has_value(product_characteristics(P, premium), R)

D(own_risk, P, R)
variable_has_value(product_characteristics(P, own_risk), R)

D(terms, P, T)
variable_has_value(product_characteristics(P, terms_type, T)

6.4. Example Simulation Trace

By applying the dynamic relations presented in Section 5 to the case study presented in this section, a number of simulation experiments have been performed. To this end, the LEADSTO simulation tool (see [2] for details) has been used. This piece of software takes as input a set of executable dynamic properties, and uses these to generate traces describing the course of events over time. Figure 6 shows the resulting simulation trace of an example scenario of client John Smith. Here, time is on the horizontal axis, and the state properties are on the vertical axis. A dark box on top of the line indicates

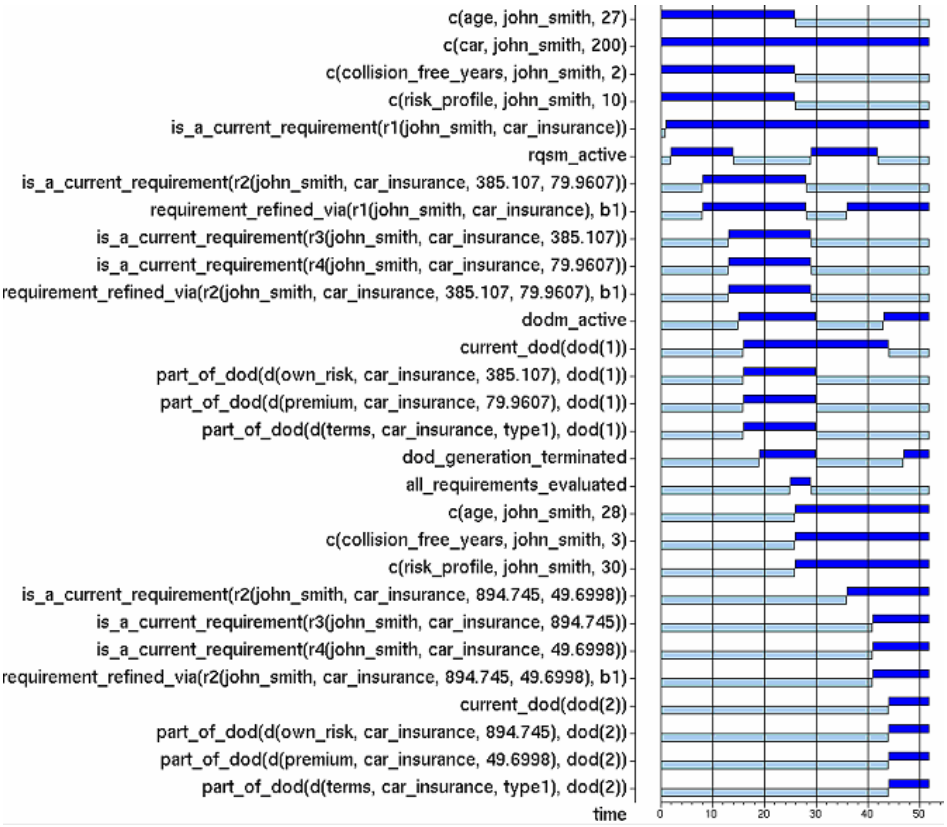

Figure 6. Example simulation trace

that a state property is true during a certain time period, and a lighter box below the line indicates that a property is false. In the scenario, there are two possible risk profiles, named profile1 and profile2. The first profile is a low-risk characteristic, meaning that the person is not willing to accept own risk and willing to accept a high premium. The second profile is more risky, meaning that the person accepts own risk and a low premium. The car type is this scenario is of type 1, a Cadillac. The type of car, the age and the number of collision free years determine the risk/premium curve (see Figure 5). Combined with the risk profile of the client, the optimal premium and own risk can be determined.

The client is this scenario has four characteristics: his age is 27, he owns a car with 200 HP, he has had two collision free years, and he has a risk profile of 10 (e.g. c(risk_profile, john_smith, 10)). Initially, requirement R1, to find a suitable insurance, is identified (see is_a_current_requirement(R1(john_smith, car_insurance)) at time point 1). Next, this requirement is refined to subrequirement R2 (see property is_a_current_requirement(R2(john_smith, car_insurance, 385.107, 79.9607)) at time point 8). Requirement R2, finding a suitable premium and own risk, can be refined in requirements R3 an R4. The risk/premium curve is calculated using the characteristics of the client (see formula (1)). Together with the risk profile, the optimal own risk (requirement R3) and the optimal premium (requirement R4) can be determined: 385.107 and 79.9607 respectively.

Next, the design objects can be generated by adding the best object characteristics that match the requirements to the current DOD. As shown in Figure 6, these variables are indeed assigned to the first DOD (e.g. part_of_dod(d(own_risk, car_insurance, 385.107), dod(1)) at time point 16). When all requirements are evaluated, the process of generating design object characteristics stops. The following year, when the client turned 28 and has had three collision free years, he increases his risk profile to 30 (see property c(risk_profile, john_smith, 30)). This increases the optimal own risk and decreases the optimal premium. The refinement of requirements starts over, with a new requirement for own risk: 894.745 and for the premium: 49.6998. Since there are new current requirements, the process of generating design object characteristics starts again, new objects are added to the second DOD, and the requirements are evaluated.

The case study shows that the model, instantiated with specific knowledge from the domain of car insurances, indeed outputs tailored car insurance proposals. As the

characteristics of the client change over time, so does the proposal for a suitable car insurance.

7. Discussion

Traditionally, product models often have been forced in a static format, entailing rigidity of two types: (1) rigidity in the manner most products are initially fit to clients and (2) rigidity in the way they are maintained over time after a client has accepted them. When offering the product via an interactive medium like Internet, as a Web service, solving such rigidity problems increases the service's success probability. To cope with these problems, this paper proposes a dynamic approach to product models. This approach supports an ongoing interaction process between the designer and the client. During such an interaction process, the background and the desires of the client may change continuously, which will lead to continuous adaptation of the product.

Globally, the approach is based on the Global Design Model (GDM) by [5]. It involves four important types of dynamical relations, i.e., (1) to identify requirements and to refine them into more specific requirements, (2) to relate specific requirements to basic design object attributes, (3) to determine derivable design object attributes from basic design object attributes, and (4) to evaluate a design object with respect to requirements. The approach has been formalised in the high-level executable language LEADSTO [2], which is a sublanguage of the predicate logical language TTL [3]. The resulting (generic) specification has been tested in a case study in the domain of car insurances.

In the domain of web services, extensive research has been conducted focusing on the composition of web services (see e.g. [12, 13, 14]). Furthermore, the specification of the precise services offered has been investigated as well, proposing for instance semantic web technology (see e.g. [11]). An additional important aspect for businesses wanting to offer a service on the web is to make this service as attractive as possible for a client, which is addressed in this paper. In [6] such tailoring is also addressed; that paper introduces a language that allows for the specification of service level agreements in a flexible, individual way. The approach also introduces a way to manage such service levels throughout the entire life cycle of such a service level agreement. The approach presented in this paper is however more generic, since it focuses on tailoring of products in general, not only particular aspects of such a product.

In future work, more complex cases will be addressed, including more realistic examples that involve a higher amount of parameters. Moreover, to evaluate its generality, the approach will be tested in other domains, such as the design of mortgages.

8. References

[1] Barringer, H., M. Fisher, D. Gabbay, R. Owens, and M. Reynolds (1996). The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons.

[2] Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J. (2005). LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T., et al. (eds.), *Proc. of the Third German Conference on Multi-Agent System Technologies, MATES'05*. LNAI, vol. 3550. Springer Verlag, pp. 165-178.

[3] Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J. (2006). Specification and Verification of Dynamics in Cognitive Agent Models. In: Nishida, T., Klusch, M., Sycara, K., Yokoo, M., Liu, J., Wah, B., Cheung, W., and Cheung, Y.-M. (eds.), *Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06*. IEEE Computer Society Press, 2006, pp. 247-254.

[4] Bosse, T., Jonker, C.M., and Treur, J. (2004). Analysis of Design Process Dynamics. In: Lopez de Mantaras, R. and Saitta, L. (eds.), *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04*, IOS Press, 2004, pp. 293-297.

[5] Brazier F.M.T., Langen P.H.G. van, and Treur J. (1996). A logical theory of design. In: J.S. Gero (ed.), *Advances in Formal Design Methods for CAD, Proc. of the Second International Workshop on Formal Methods in Design*. Chapman & Hall, New York, 1996, pp. 243-266.

[6] Dan, A., *et al.* (2004). Web Services on Demand: WSLA Driven Automated Management. *IBM Systems Journal*, vol. 43, 2004, pp. 136-158.

[7] Dardenne, A., Lamsweerde, A. van, and Fickas, S. (1993). Goal-directed Requirements Acquisition. *Science in Computer Programming*, vol. 20 (1993), pp. 3-50.

[8] Galton, A. (2003). Temporal Logic. *Stanford Encyclopedia of Philosophy*, URL: http://plato.stanford.edu/entries/logic-temporal/#2

[9] Galton, A. (2006). Operators vs Arguments: The Ins and Outs of Reification. *Synthese*, vol. 150 (2006), pp. 415-441.

[10] Manzano, M., (1996). Extensions of First Order Logic, Cambridge University Press, 1996.

[11] McIlraith, S.A., Son, T.C., and Zeng, H. (2001). Semantic Web services. IEEE Intelligent Systems, vol. 16, 2001, pp. 46-53.

[12] Narayanan, S., and McIlraith, S.A. (2002). Simulation, Verification and Automated Composition of Web Services. In: Proceedings of the 11th International Conference on World Wide Web, ACM Press, 2002, pp. 77-88.

[13] Srivastava, B., and Koehler, J. (2003). Web Service Composition – Current Solutions and Open Problems. In: Proceedings of the Workshop on Planning for Webs Services, 2003.

[14] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. (2003). Quality Driven Webs Service Composition. In: Proceedings of the 12th International conference on World Wide Web, ACM Press, 2003, pp. 411-421.