Analyzing Variability of Software Product Lines Using Semantic and Ontological Considerations

Iris Reinhartz-Berger[1], Nili Itzik[1], and Yair Wand[2]

[1] Department of Information Systems, University of Haifa, Israel
iris@is.haifa.ac.il, nitzik@campus.haifa.ac.il,
[2] Sauder School of Business, University of British Columbia, Canada
yair.wand@ubc.ca

Abstract. Software Product Line Engineering (SPLE) is an approach to systematically reuse software-related artifacts among different, yet similar, software products. Previewing requirements as drivers of different development methods and activities, several studies have suggested using requirements specifications to identify and analyze commonality and variability of software products. These studies mainly employ semantic text similarity techniques. As a result, they might be limited in their ability to analyze the variability of the *expected behaviors* of software systems as perceived from an external point of view. Such a view is important when reaching different reuse decisions. In this paper we propose to introduce considerations which reflect the behavior of software products as manifested in requirement statements. To model these behavioral aspects of software requirements we use terms adapted from Bunge's ontological model. The suggested approach automatically extracts the initial state, external events, and final state of software behavior. Then, variability is analyzed based on that view.

Keywords: Software Product Line Engineering, Variability analysis, Requirements Specifications, Ontology.

1 Introduction

Software Product Line Engineering (SPLE) is an approach to systematically reuse software-related artifacts among different, yet similar, software products [6], [19]. Reuse of artifacts, such as requirements specifications, design documents and code, often results in the creation of a myriad of variants. Managing such a variety of artifacts' variants is a significant challenge. Thus, SPLE promotes the definition and management of software product lines (SPLs), which are families of similar software systems, termed software products. An important aspect of SPLE is managing the *variability* that exists between the members of the same SPL. In this context, variability is defined as "the ability of an asset to be efficiently extended, changed, customized, or configured for use in a particular context" [10].

In SPLE, different artifacts need to be managed. Of those, requirements management is of special interest due to several reasons. First, requirements represent the expectations of different stakeholders from the requested system. These stakeholders

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 150–164, 2014.
© Springer International Publishing Switzerland 2014

include users and customers and not just developers. Second, requirements are the drivers of other development activities, including analysis, design, implementation, and testing. Finally, requirements are relevant to many development methods, including agile ones (through concepts such as user stories).

Several studies have suggested using requirements specifications in order to identify and analyze commonality and variability of software products. In these studies, requirements are operationalized or realized by features, and variability is usually represented as feature diagrams, the main aid for representing and managing variability in SPLE [5], [11]. The current studies commonly apply only semantic similarity metrics, that is, seek similarities of terminology, in order to identify common features, create feature diagrams, and analyze the variability of the resultant feature diagrams. As we will show via examples, using only semantic considerations might limit the ability to analyze the variability of the *expected behaviors* of software systems as perceived from an external point of view of a user or a customer. Such a view is important for reaching different reuse decisions, e.g., when conducting feasibility studies, estimating software development efforts, or adopting SPLE. In addition, current variability analysis methods take into account intermediate outcomes of the behavior that may not matter to external stakeholders, such as users and customers. For example, a system may intermediately keep information in case a transaction fails, but this would be of no interest when the behavior ends successfully. Hence, when analyzing variability of software products, we aim at minimizing the impact of intermediate outcomes which cannot be used for (and might confound) comparing the products from an external point of view.

In this work we propose to overcome the shortcomings of pure semantic-based variability analysis by combining semantic similarity with similarity of software behavior as manifested in requirement statements. To compare software behavior we apply an ontological view of dynamic aspects of systems which we proposed in earlier work [20], [21]. For a given requirement, we consider the behavior it represents in the application ("business") domain. Taking an external point of view, behavior is described in terms of the initial state of the system before the behavior occurs, the external events that trigger the behavior, and the final state of the system after the behavior occurs. We use semantic metrics to evaluate the similarity of related behavioral elements and use this similarity to analyze variability.

The rest of this paper is structured as follows. Section 2 reviews related work, exemplifying limitations of current approaches. Section 3 briefly provides the ontological background and our framework for classifying software variability. Section 4 introduces the ontological approach to variability analysis and demonstrates its applicability. Section 5 presents preliminary results and discusses advantages and limitations. Finally, Section 6 summaries the work and presents future research directions.

2 Related Work

As mentioned above, the main approach to variability analysis in SPLE is semantic – based on text similarity measures. Semantic text similarity measures are commonly classified as knowledge-based or corpus-based [9], [16].

Corpus-based measures identify the degree of similarity based on information derived from large corpora (e.g., [4], [12], and [22]). Latent Semantic Analysis (LSA) [12], for example, is a well-known method that analyzes the statistical relationships among words in a large corpus of text. Sentence similarity is computed as the cosine of the angle between the vectors representing the sentences' words.

Knowledge-based measures use information drawn from semantic networks. Many of these methods use WordNet [26] for measuring word (or concept) similarity. This can be done in different ways, including measuring path length between terms on the semantic net or using information content, namely, the probability to find the concept in a given net. Several measures have been suggested to extend word similarity to sentence similarity. These measures consider sentences as vectors, sets, or lists of words and suggest ways to calculate sentence similarity using word similarities (e.g., [13], [14], and [16]). The MCS method [16], for example, calculates sentence similarity by finding the maximum word similarity score for each word in a sentence with words in the same part of speech class in another sentence. The derived word similarity scores are weighted with the inverse document frequency scores that belong to the corresponding word.

In the context of analyzing software products variability, different studies have suggested ways to use textual requirements to generate variability models in general and feature diagrams in particular. Examples of such studies are [7], [17], and [25]. In [25], for instance, the semantic similarity of the requirements is measured using LSA. Then, similar requirements are grouped using a hierarchical agglomerative clustering algorithm. Finally, a Requirements Description Language enables the specification and composition of variant features.

All the above methods employ only semantic considerations. Furthermore, the similarity calculation takes into consideration the full text of a requirement statement. As mentioned before, such statements might include aspects (e.g., intermediate outcomes) that are less or not relevant for analyzing variability from an external perspective. We illustrate the limitations of the current methods and motivate our approach, using a series of examples.

The first example refers to the following requirements:

(1) "The system should be able to report on any user update activities";
(2) "Any user should be able to report system activities".

Applying the well-known and commonly used semantic similarity method LSA[1], the similarity of these sentences is 1. This would imply that their semantic meanings are identical, and hence no variability between these requirements exists. It is clear, however, that these requirements are quite different: the first represents behavior that is internal and likely aims at detecting suspicious user update activities. The second requirement represents a behavior triggered by an external user who intends to report his/her system activities.

As a second example, consider the following two requirements:

(3) "The system will allow different functions based on predefined user profiles";
(4) "Different operations should be allowed for different user profiles".

---

[1] We used LSA implementation that can be accessed via `http://lsa.colorado.edu/`.

In this case, LSA results with a low similarity value of 0.38, failing to reflect the situation accurately: the two requirements represent very similar domain behaviors.

Finally, the following two requirements can be considered similar from an external point of view, although they differ in their levels of details of intermediate actions.
(5) “When the client activates an activity she is allowed to perform, the system displays the outputs of the activity.”
(6) “If the user is authorized to perform an action, the system initializes the parameters needed by the action. The user performs the action and the system responds that the action was performed. Finally, the user requests to display the outputs, and the system presents the action's outcomes.”

However, the LSA-based value of the similarity of these two requirements is relatively low (0.57), failing to reflect their similarity from an external point of view.

To overcome the above limitations, we propose to combine a semantic approach and considerations which reflect system behavior as manifested in requirements statements and modeled ontologically.

3 Bung’s Ontology and Software Variability Classification

We use concepts from Bunge's ontological model [2, 3] and its adaptations to software and information systems [23, 24] in order to define behaviors and use them for variability analysis. We have chosen this ontology because it formalizes concepts that are important for representing functionality and behaviors. Specifically, these concepts include things, states, events, and transformations. Furthermore, Bunge’s ontological model has already served us to define software variability classes [20, 21].

Bunge's ontological model [2], [3] describes the world as made of *things* that possess *properties*. Properties are known via *attributes*, which are characteristics assigned to things by humans. A *state variable* is a function which assigns a value to an attribute of a thing at a given time. The *state* of a thing is the vector of state variables’ values at a particular point in time. For a state s, s.x denotes the value of the state variable x in s. An *event* is a change of a state of a thing. An event can be external or internal. An *external event* is a change in the state of a thing as a result of an action of another thing. An *internal event* is a change which arises due to an internal transformation in the thing. Finally, a state can be stable or unstable: a *stable state* can be changed only by an external event. An *unstable state* will be changed by an internal event.

We exemplify the above concepts using a library management domain. In this domain, *book status* can be considered a state variable, defining whether a book is borrowed, on the shelf, or in repair; *ready to lend* can be considered a stable state, when it can accept the external event – *borrow book* (generated by a reader); and *book becomes past-due* can be considered an internal event, which is initiated when a certain period has passed from borrowing and the book is not yet returned.

Using these concepts, we defined in [20] *a behavior* as a triplet (s_1, E, s^*). s_1 is termed the *initial state* of the behavior and s^* – the *final state* of the behavior. We assume that the system can respond to external events when in s_1 (i.e., s_1 is an input sensitive state, see [20], [21]). s^* is the first stable state the thing (the real domain or a system) reaches when it starts in state s_1 and the external events sequence E=<e_1,…,

e_n> occurs. The full behavior includes intermediate states the thing traverses due to its own transformations in response to the external events. However, only (s_1, E, s*) are "visible" from an external (user) point of view.

In our example, *borrowing* can be considered a behavior, which starts in the state *ready to lend* (i.e., when the book status is "on the shelf" and the librarian is "available"), is triggered by the external event *borrow book,* and ends in the state *book is borrowed* (i.e., the book status is "borrowed" and the librarian is "available" again).

We further made two assumptions regarding the things whose behavior we model [21]: *no interruption* (external events can affect a thing only when it or at least one of its components is in a stable state) and *stability assumption* (all things we deal with in practice will eventually reach stable states).

Finally, we defined similarity of behaviors in terms of similarity of their external events and states [20]:

Event similarity: Two external events are considered similar if they appear to be the same in the application domain.

State similarity: Two states s and t are considered similar with respect to a set of state variables X, iff $\forall x \in X$ s.x = t.x. X is termed the *view of interest.*

Based on these definitions we identified eight classes of external variability, namely, variability that refers to software *functionality* as visible to users (see Table 1).

Table 1. External variability classes based on systems' behaviors

#	s_1	E	s^*	Class Name
1.	*similar*	*similar*	*similar*	Completely similar behaviors
2.	*similar*	*not similar*	*similar*	Similar cases and responses, different interactions
3.	*similar*	*similar*	*not similar*	Similar triggers, different responses
4.	*similar*	*not similar*	*not similar*	Similar cases, different behaviors
5.	*not similar*	*similar*	*similar*	Different cases, similar behaviors
6.	*not similar*	*not similar*	*similar*	Different triggers, similar responses
7.	*not similar*	*similar*	*not similar*	Different cases and responses, similar interactions
8.	*not similar*	*not similar*	*not similar*	Completely different behaviors

In the current work, we use textual software requirements as the basis for automatic identification of domain behaviors and their elements (namely, the initial and final states and the external events). We use semantic measurements in order to refine event and state similarity definitions.

4 Deriving Domain Behaviors from Software Requirements

Perceiving a software system as a set of intended changes in a given domain, we focus on systems' behaviors as specified by or represented in functional requirements. Functional requirements commonly refer to *actions* (what should be performed?) and *objects* (on what objects, also termed *patients*, should the action be performed?). They can further refer to the *agents* (who performs the action?), the *instruments* (how the action is performed?), and the *temporal constraints* (when is the action preformed? in what conditions is it performed?).

There are different ways to write and phrase functional requirements. For our purpose, we assume that they are specified as *user stories* or *descriptions of use cases*.

We further assume that each use case or user story represents a *single behavior* of the requested system[2]. For example, consider the following requirement which describes a typical use case in a library management system:

> When the home page is displayed, a borrower borrows a book copy by herself. She enters the copy identification number after she provides the borrower number. If the copy identification number and the borrower number are valid, the system updates the number of available copies of that title.

Our approach consists of four steps: (1) *pre-processing* which checks the quality of the individual requirements and identifies the need for corrections or improvements; (2) *extraction of the main behavioral elements* from a requirement, e.g., the requirement's agents (who?), actions (what?), and patients (on what objects?); (3) *Classification* of the extracted main behavioral elements according to the ontological definition of behavior (in terms of states and events); and (4) *measuring requirements variability* based on the framework presented in [20], [21].

Pre-processing is out of the scope of this paper. It may use existing quality models, such as that presented in [1]. In the following sub-sections we elaborate on steps 2-4.

4.1 Extraction of the Main Behavioral Elements

In order to extract the main behavioral elements of software requirements we use semantic role labeling (SRL) [8]. This approach labels constituents of a phrase with their semantic roles in the phrase. Currently, we refer to five semantic roles which are of special importance to functional requirements. These roles, their labels, and the aspects they fulfill in functional requirements are listed in Table 2.

Table 2. The semantic roles we use in our work

Label	Role	Assigned to	Aspects fulfilled in requirements
A0	Agent	Agents, causers, or experiencers	Who?
A1	Patient	Undergoing state change or being affected by the action	On what?
A2	Instrument	Instruments, benefactives, attributes	How?
AM-TMP	Temporal modifier	Time indicators specifying when an action took place	When?
AM-ADV	Adverbial modifier	Temporally related (modifiers of events), intentional (modifiers of propositions), focus-sensitive (e.g., only and even), or sentential (evaluative, attitudinal, viewpoint, performative)	In what conditions?

Using SRL[3], we specify for each requirement R a list of behavioral vectors $BV_R=\{bv_i\}_{i=1..n}$. Two types of behavioral vectors are identified: action and non-action vectors. The following definitions formally specify the behavioral vectors for these two types. Examples are provided immediately afterwards.

---

[2] If this is not the case, a pre-processing done by a requirements engineer is needed to split the requirements statements to separate expected behaviors.

[3] We specifically use the system at `http://barbar.cs.lth.se:8081/` or `http://en.sempar.ims.uni¬stuttgart.de/`.

Definition 1. An *action vector* represents an activity (identified by a verb) in the behavior: $bv_i ::= (Agent_i, Action_i, Patient_i, Instrument_i, Source_i)$, where:

- $Agent_i$, $Patient_i$ and $Instrument_i$ are as explained in Table 2.
- $Action_i$ is the verb predicate of the phrase.
- $Source_i \in \{AM\text{-}TMP, AM\text{-}ADV, None\}$ indicates whether the vector originates from a modifier (temporal or adverbial) or a non-modifier phrase, respectively.

An action vector is derived from a non-modifier phrase or a compound modifier phrase that is further parsed to reveal its constituting components (e.g., agents and actions).

Definition 2. A *non-action vector* represents the temporal or adverbial pre-condition of the behavior (or part of it): $bv_i ::= (Modifier_i, Source_i)$, where:

- $Modifier_i$ includes the atomic modifier phrase
- $Source_i \in \{AM\text{-}TMP, AM\text{-}ADV\}$ indicates whether the vector originates from a temporal or adverbial modifier, respectively.

A non-action vector is derived from an atomic modifier phrase which includes no verb (and thus is not further parsed by SRL).

Table 3 lists the derived behavioral vectors for our previous requirement of the library management example. Vector #5 is a non-action vector. All other vectors represent actions. We further replace pronouns with their anaphors (i.e., the nouns to which they refer) using the algorithm in [18] (e.g. the agent "she" becomes "a borrower").

Table 3. Examples of behavioral vectors

#	Agent	Action	Patient	Instrument	Modifier	Source
1	[4]	is displayed	the home page			AM-TMP
2	a borrower	Borrows	a book copy	by herself		None
3	[5]~~She~~ [a borrower]	Enters	the copy identification number			None
4	[5]~~She~~ [a borrower]	Provides	the borrower number			AM-TMP
5	[4]				the copy identification number and the borrower number are valid	AM-ADV
6	The system	Updates	the number of available copies of that title			None

The next step in the analysis is to arrange the behavioral vectors of each requirement in a temporal order. We do this by constructing temporal graphs:

Definition 3. Given a requirement R and its derived list of behavioral vectors BV_R, the *temporal graph* is defined as $TG_R=(BV_R, E)$, where $e=(bv_1, bv_2)\in E$ implies that $bv_1, bv_2 \in BV_R$ and bv_1 temporally precedes bv_2 (notation: $bv_1 \dashrightarrow bv_2$).

The construction of edges in this graph is done in two steps. First, we use *syntactic ordering*, based on the order of the argument vectors in the requirement's phrasing.

[4] Note that since we are interested in automated analysis, we cannot incorporate here assumptions about what causes these actions (e.g., the system or an external user).

[5] Replacement of a pronoun by the relevant noun is indicated with ~~pronoun~~ [noun].

Second, we apply *semantic ordering*, using the machine learning algorithm suggested in [15], to update the syntactic edges based on six types of temporal relations derived from the text. These relations are listed in Table 4. Whenever a semantic relationship contradicts a syntactic one, we use the semantic relationship as shown in the table.

Table 4. The temporal relations for overriding syntactic edges with semantic ones; W, X, Y, and Z are temporal phrases or events, ⇢ indicates their order

The graph after phase 1	**Detected semantic temporal relation**	**The graph after phase 2**
W ⇢ X ⇢ Y ⇢ Z	Y before X Y ibefore X	W ⇢ Y ⇢ X ⇢ Z
W ⇢ X ⇢ Y ⇢ Z	X begins Y[6] X ends Y[6] X includes Y[6] X simultaneous Y[6]	X W Z Y

Fig. 1 exhibits the temporal graph for our example (Table 3). The changes the semantic ordering causes to the syntactic order (the gray arrows) are depicted with the black arrows[7].

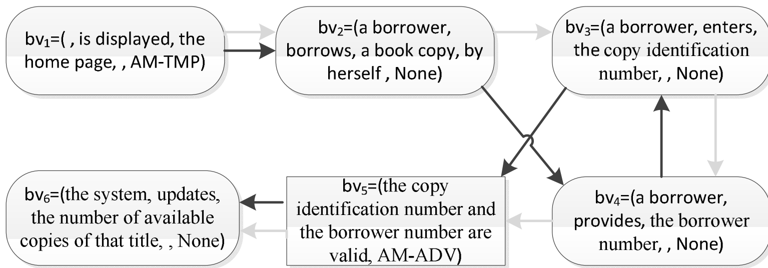

Fig. 1. The temporal graph generated for our example; Ellipses represent action vectors and rectangles – non-action vectors

4.2 Classification of the Behavioral Vectors

We now turn to the classification of the dynamic aspects of the requirements to initial states, external events, and final states. To this end, we first classify each behavioral vector into external, internal, or unknown (with respect to the requested system). In particular, we examine the Agent and Action components of action vectors: the agent can be *internal*, *external*, or *missing* (as in passive phrases)[8]; independently, the action

[6] In all these cases X and Y are executed in parallel (at least partially).

[7] We assume that the requirements are well-written (i.e., include no ambiguities and contradictions) after the pre-processing step. Thus, the temporal graph is a directed acyclic graph.

[8] We maintain a list of terms representing internal agents, including: “the system”, “the application”, and the explicit name and abbreviation of the requested system. The requirements analyst may update this default list to include the main components of the requested system.

can have an *active* or a *passive* meaning[9]. All non-action vectors are considered internal, as they do not represent an actual action, but a pre-condition for the behavior (or part of it). Accordingly, we identify six generic cases (see Table 5).

Table 5. Classifying behavioral vectors into internal, external, and unknown

Case	**Classification** class(bv)	**Example**	**Comments**
1. An **action vector** with an **external agent** and an **active meaning**	EXTERNAL	"a borrower borrows a book"	The action is performed by an external agent
2. An **action vector** with an **external agent** and a **passive meaning**	INTERNAL	"a borrower receives an email message"	The system acted on an external agent
3. An **action vector** with an **internal agent** and an **active meaning**	INTERNAL	"the system updates the number of available copies"	The action is performed by the system
4. An **action vector** with an **internal agent** and a **passive meaning**	EXTERNAL	"the system receives the number of available copies"	The system is acted on
5. An **action vector** with a **missing agent**	UNKNOWN	"a book is borrowed"	The vector cannot be deterministically classified
6. A **non-action vector**	INTERNAL	if "the book copy is valid" or if "the borrower is new"	The vector represents a pre-condition for the behavior (or part of it)

Returning to our example (Table 3): class(bv_1) = UNKNOWN; class(bv_2) = class(bv_3) = class(bv_4) = EXTERNAL; class(bv_5) = class(bv_6) = INTERNAL.

Behavioral vectors classified as EXTERNAL represent actions performed by external agents and therefore are considered ***external events (E)***. In contrast, behavioral vectors classified as INTERNAL represent actions performed or pre-conditions checked by the system. They are considered to reflect states: initial, final, or intermediate. As an initial state describes the state *before* the behavior occurs, only internal behavioral vectors that *precede* (in the temporal graph) the sequence of external behavioral vectors will be taken into consideration for defining the initial state of the behavior. Of those, only vectors whose sources are modifiers (and thus represent pre-conditions) are considered the ***initial state of the behavior*** (s_1). Following similar arguments, only internal behavioral vectors which *follow* the sequence of external behavioral vectors will be taken into consideration for defining the final state of the behavior. Of those, only action vectors whose sources are not modifiers (and thus represent actual internal actions) are considered the ***final state of the behavior*** (s^*). All other internal behavioral vectors, i.e., those interleaved with the external behavioral vectors, are considered to be manifested by intermediate states. Such actions (and related states) are not currently taken into consideration in our analysis, which is based on an external view of behaviors.

[9] Note that passive actions can use an active form of the verb (e.g., "receive" and "get"). Thus, we maintain a list of such verbs.

Behavioral vectors for which the agent is unknown are classified at this stage into multiple behavioral elements (e.g., both initial state and external events). The decision whether these vectors represent internal or external actions is taken in a later stage, when calculating for each vector the most similar counterparts.

We next formally define the behavior associated with a requirement and exemplify this definition on our requirement:

Definition 4. Given a requirement R, its derived list of behavioral vectors BV_R, and its temporal graph TG_R, the *behavior associated with* R is defined as a triplet $B_R=(s_1, E, s^*)$, where:

- The *initial state* (s_1) includes all internal or unknown vectors originated from modifiers (i.e., may represent pre-conditions) and *precede* all external vectors in the temporal graph representing the behavior. Formally expressed,
  $s_1 = \{bv \in BV_R \mid class(bv) \in \{INTERNAL, UNKNOWN\}$ and $bv.Source \in \{AM\text{-}TMP, AM\text{-}ADV\}$ and $\neg\exists path\ p \in TG_R$ such that $bv' \dashrightarrow bv \in p$ and $class(bv') = EXTERNAL\}$
- The *external events (E)* include all potentially external behavioral vectors (namely, external vectors and action vectors with unknown agents). Formally expressed,
  $E = \{bv \in BV_R \mid class(bv) \in \{EXTERNAL, UNKNOWN\}\}$
- The *final state (s*)* includes all internal or unknown vectors that do not originate from modifiers (i.e., may represent actual actions) and *follow* all external vectors in the temporal graph representing the behavior. Formally expressed,
  $s^* = \{bv \in BV_R \mid class(bv) \in \{INTERNAL, UNKNOWN\}$ and $bv.Source = None$ and $\neg\exists path\ p \in TG_R$ such that $bv \dashrightarrow bv' \in p$ and $class(bv') = EXTERNAL\}$

In our previous example, we obtain the classification of behavioral vectors as shown in Table 6. Note that bv_1 appears twice as its agent is unknown and hence it can be considered either an external event or an initial state. bv_5 = (the copy identification number and the borrower number are valid, AM-ADV) does not appear at all as it represents a pre-condition originated from an adverbial modifier and appearing after external events. Thus, bv_5 cannot be considered an initial neither final state (but rather an intermediate state).

Table 6. An example of the outcome of the behavioral vectors classification phase

S_1 (initial state)	E (external event to which the system responds)	S* (final state the system is expected to have)
bv_1=(, is displayed, the home page, , AM-TMP)	bv_1=(, is displayed, the home page, , AM-TMP) bv_2=(a borrower, borrows, a book copy, by herself , None) bv_4=(a borrower, provides, the borrower number, , None) bv_3=(a borrower, enters, the copy identification number, , None)	bv_6=(the system, updates, the number of available copies of that title, , None)

4.3 Measuring Requirements Variability

Having two requirements, their behavioral vectors, and the classification of the vectors to initial states, external events, and final states, we now define *behavioral similarity*. The definitions are followed by an example.

Definition 5 (Behavioral Vectors Similarity). Given two behavioral vectors, the vectors similarity is calculated as follows:

1. If the two vectors are *action vectors*, the vectors similarity is the weighted average of their component semantic similarities. Formally expressed,
   $VS\ (v_1,\ v_2)\ =\ \frac{\sum_{comp \in \{Agent, Action, Patient, Instroment\}} \delta_{comp} * w_{comp} * sim(v_1.comp, v_2.comp)}{\sum_{comp \in \{Agent, Action, Patient, Instroment\}} \delta_{comp} * w_{comp}}$,
   where:
   - w_{comp} is the weight given to a specific vector component (agent, action, patient, or instrument), $\sum_{comp \in \{Agent, Action, Patient, Instroment\}} w_{comp} = 1$.
   - δ_{comp} is 1 if the component comp exists (i.e., it is not empty in both v_1 and v_2) and 0 otherwise.
   - $sim(v_1.comp,\ v_2.comp)$ is the semantic similarity of the two vectors' components.
2. If the two vectors are *non-action* vectors, the vectors similarity is the semantic similarity of their modifier components. Formally expressed,
   $VS\ (v_1, v_2) = sim(v_1.Modifier, v_2.Modifier)$, where
   - $sim(v_1.Modifier, v_2.Modifier)$ is the semantic similarity of the modifier components of the two vectors.
3. If one vector is an *action vector* (say v_1) and the other is a *non-action vector*, the vectors similarity is the semantic similarity between the corresponding phrases. Formally expressed, $VS\ (v_1, v_2) = sim(v_1, v_2.Modifier)$, where:
   - $sim(v_1, v_2. Modifier)$ is the semantic similarity between the concatenation of the agent, action, patient, and instrument components of the first vector and the modifier component of the second vector.

Definition 6 (Behavioral Element Similarity). Given two requirements, R_1 and R_2, and their behavioral vectors that are classified as the same element bh (initial state, external events, or final state), the behavioral element similarity is calculated as the average of the maximal pair-wise similarities. Formally expressed:

$$BS\ (R_1, R_2 \mid bh) = \begin{cases} 0 & R_1.\mathrm{bh} \neq \emptyset \ and\ R_2.\mathrm{bh} = \emptyset \\ \frac{\sum_{i=1}^{n} \max_{j=1..m} VS(v_i, v'_j)}{n} & R_1.\mathrm{bh} = \{v_1, \ldots, v_n\} \text{ and } R_2.\mathrm{bh} = \{v'_1, \ldots, v'_m\}, \\ 1 & R_1.\mathrm{bh} = \emptyset \end{cases}$$

where:

- $R_1.\mathrm{bh}, R_2.\mathrm{bh}$ are the behavioral vectors classified as the element bh in requirements R_1 and R_2, respectively; $R_i.\mathrm{bh}=\emptyset$ means that no behavioral vectors were classified as bh.
- $VS\ (v_i, v'_j)$ is the behavioral vector similarity of v_i and v'_j.

As an example consider the following requirements:

1.	"When a borrower borrows a book copy by herself, she enters the copy identification number and the borrower number. The system updates the number of available copies of that title."
2.	"When a librarian lends a book copy to a borrower, she enters the copy identification number and the borrower number. The system updates the number of available copies of that title and stores the lending details (when, by whom, to whom)."

For calculating component semantic similarities we used an MCS version that handles phrases rather than complete sentences. We set the component weights to 0.3,

0.4, 0.2, and 0.1 for agents, actions, patients, and instruments, respectively, perceiving agents and actions as the dominant components in behavioral vectors similarities. We obtain initial state similarity for the given requirements of 1 (no special pre-conditions in both requirements), external events similarity of 0.78 (due to differences in the agents that initiate the events), and final state similarity of 1 (as the final state of the first requirement is included in the final state of the second requirement). Note that we chose an asymmetric metric for defining behavioral element similarity, meaning that BS (R_1, R_2 | bh) ≠ BS (R_2, R_1 | bh), as we perceive similarity as the ability to reuse behavior R_2 when behavior R_1 is required. The asymmetry in this measure reflects the possibility that it might be acceptable to substitute one behavior for another, but not the second for the first, as exemplified by the two requirements above.

Based on the behavioral element similarity, we classify the outcome of comparing each pair of requirements to one of the eight variability classes in Table 1. To this end, we define *event similarity threshold* (th_e) and *state similarity threshold* (th_s):

1. Initial states are considered similar if and only if BS (R_1, R_2 | s_1) > th_s.
2. External events are considered similar if and only if BS (R_1, R_2 | E) > th_e.
3. Final states are considered similar if and only if BS (R_1, R_2 | s*) > th_s.

Assuming an event similarity threshold greater than 0.5 (e.g., 0.8), the variability class to which requirement 1 belongs with respect to requirement 2 is # 2 (see Table 1: similar cases and responses, different interactions). This class accurately describes the requirements variability.

5 Preliminary Results

To evaluate the proposed approach, we compared its outcomes to evaluations by experts. We provided five experts, each having 10 to 25 years of experience in requirements engineering and software development, with 10 requirements. For each requirement, four alternative systems to be considered were presented to the experts. Each alternative was describes as a requirement. The full set of requirements and alternatives is discussed in [20][10]. We asked the experts to rank the four alternatives for each requirement based on the similarity to the given requirement in terms of the amount of changes needed to adapt the alternatives to the requirement. Since experts' ranking requires some subjective considerations, there was no full agreement between the experts regarding the ranking of the alternatives. Therefore, we defined for each pair of alternatives, Si and Sj (i, j=1…4, i>j), relation "Si is not better than Sj" (Si≤Sj). For each requirement there were four possible alternatives yielding six such relations. This provided a total of 60 relations for the 10 requirements. There were 55 relations on which most experts (at least four out of the five experts, 80%) agreed.

We conducted the analysis described in this work for the same set of requirements. We used the behavioral element similarities to calculate overall similarity, which can serve as a basis for ranking alternatives. The weight of initial state similarity was set to 0.2, the weight of external events – 0.3, and the weight of final state similarity – 0.5. This reflected an assumption that the final state of behaviors (usually specifying

[10] It can be accessed at `http://mis.hevra.haifa.ac.il/~iris/research/OA/QuestionnaireEng.pdf`.

system output) is the dominant element in defining behavior similarity. We followed a similar procedure using the well-known semantic similarity method LSA, which as noted is based only on semantic considerations. Table 7 summarizes the results of the ontological approach and LSA with respect to experts[11].

Table 7. Comparing the results of the ontological approach and LSA with experts

	Ontological approach	**LSA**
Number of experts' relations found by the method (out of 55)	51 relations (93%)	45 relations (82%)

As can be seen, our approach performed better than LSA in comparison to rankings by experts. We believe that our approach has an additional advantage to better performance – it is self-explanatory. Users of the approach, who are expected to be requirements analysts, can see not only the overall calculated similarity, but also more details: initial state, external events, and final state similarities. This can help make their reuse decisions more evidence-based and feasibility studies more systematic.

Analyzing the relations missed by the ontological approach, we observed the following. First, some of the requirements included phrases that explain reasons, e.g., "so the librarian can make inter-library loans". These phrases were interpreted by the approach as an integral part of the behavior (part of the external events in this case). Second, in a few cases, where the requirements statements included very complicated sentences, SRL failed to correctly identify the agents, actions, patients and/or instruments of the different phrases. Finally, we observed that in some cases our approach resulted with the conclusion that two alternatives are very similar to the given requirement and the experts subjectively preferred one alternative over the other.

6 Summary and Future Work

We proposed a method to analyze variability and similarity of software requirements based on combining semantic and behavioral aspects of requirement statements. To formalize the external (user-oriented) aspects of software behavior we used an ontological model where a specific functional requirement is modeled as a triplet: initial state, external events, and the final state of the system. We have shown how such a representation can be obtained automatically by: (1) applying semantic analysis to requirements statements to identify behavioral vectors; (2) describing the vectors in common terms; (3) ordering the vectors temporally based on modifiers identified in the semantic analysis; and (4) extracting the initial state, external events, and final state for each functional requirement. We then suggested a way to measure the similarity of two requirements based on each element of their behavioral triplets and classified pairs of requirements to one of eight variability classes. In a preliminary evaluation, the approach yielded results more similar to experts' evaluations than those of a well-known semantic similarity measure – LSA.

[11] Elaborations can be found at
`http://mis.hevra.haifa.ac.il/~iris/research/SOVA/`.

In the future, we intend to extend the approach in several ways. First, we intend to consider additional semantic roles, e.g., location modifiers. Second, we plan to refine the similarity measures to include a choice of specific state variables rather than complete behavioral vectors, thus having a way to reflect user views more faithfully. This will enable us to analyze variability of software requirements from different points of view that may reflect different purposes or stakeholders. Users may consider two software behaviors similar while developers may consider them different, or vice versa. Similarly, such differences might exist among users. The choice of state variables to represent different points of view can be included in the behavioral analysis and hence in the similarity calculation. Third, we also intend to take into account in the variability analysis different ordering of the occurrence of external events.

References

1. Berry, D.M., Bucchiarone, A., Gnesi, S., Lami, G.,Trentanni, G.: A new quality model for natural language requirements specifications. In: The International Workshop on Requirements Engineering: Foundation of Software Quality, REFSQ (2006)
2. Bunge, M.: Treatise on Basic Philosophy. Ontology I: The Furniture of the World, vol. 3. Reidel, Boston (1977)
3. Bunge, M.: Treatise on Basic Philosophy. Ontology II: A World of Systems, vol. 4. Reidel, Boston (1979)
4. Burgess, C., Livesay, K., Lund, K.: Explorations in context space: Words, sentences, discourse. Discourse Processes 25(2-3), 211–257 (1998)
5. Chen, L., Babar, M.A.: A systematic review of evaluation of variability management approaches in software product lines. Information and Software Technology 53(4), 344–362 (2011)
6. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
7. Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M.: On-demand feature recommendations derived from mining public product descriptions. In: 33rd IEEE International Conference on Software Engineering (ICSE 2011), pp. 181–190 (2011)
8. Gildea, D., Jurafsky, D.: Automatic Labeling of Semantic Roles. Computational Linguistics 28(3), 245–288 (2002)
9. Gomaa, W.H., Fahmy, A.A.: A Survey of Text Similarity Approaches. International Journal of Computer Applications 68(13), 13–18 (2013)
10. Jaring, M.: Variability engineering as an Integral Part of the Software Product Family Development Process, Ph.D. thesis, The Netherlands (2005)
11. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) – feasibility study. Technical report no. CMU/SEI-90-TR-21). Carngie-Mellon University, Pittsburgh (1990)
12. Landauer, T.K., Foltz, P.W., Laham, D.: Introduction to Latent Semantic Analysis. Discourse Processes 25, 259–284 (1998)
13. Li, Y., McLean, D., Bandar, Z.A., O'Shea, J.D., Crockett, K.: Sentence Similarity Basedon Semantic Nets and Corpus Statistics. IEEE Transactions on Knowledge and Data Engineering 18(8), 1138–1150 (2006)

14. Malik, R., Subramaniam, V., Kaushik, S.: Automatically Selecting Answer Templates toRespond to Customer Emails. In: The International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 1659–1664 (2007)
15. Mani, I., Verhagen, M., Wellner, B., Lee, C.M., Pustejovsky, J.: Machine learning of temporal relations. In: The 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, pp. 753–760 (2006)
16. Mihalcea, R., Corley, C., Strapparava, C.: Corpus-based and knowledge-based measures of text semantic similarity. In: The 21st National Conference on Artificial Intelligence (AAAI 2006), vol. 1, pp. 775–780 (2006)
17. Niu, N., Easterbrook, S.: Extracting and modeling product line functional requirements. In: The 16th IEEE International Requirements Engineering Conference (RE 2008), pp. 155–164 (2008)
18. Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., Manning, C.: A Multi-Pass Sieve for Coreference Resolution. In: The conference on Empirical Methods in Natural Language Processing (EMNLP 2010), pp. 492–501 (2010)
19. Pohl, K., Böckle, G., van der Linden, F.: Software Product-line Engineering: Foundations, Principles, and Techniques. Springer (2005)
20. Reinhartz-Berger, I., Sturm, A., Wand, Y.: Comparing Functionality of Software Systems: An Ontological Approach. Data & Knowledge Engineering 87, 320–338 (2013)
21. Reinhartz-Berger, I., Sturm, A., Wand, Y.: External Variability of Software: Classification and Ontological Foundations. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 275–289. Springer, Heidelberg (2011)
22. Turney, P.D.: Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 491–502. Springer, Heidelberg (2001)
23. Wand, Y., Weber, R.: On the ontological expressiveness of information systems analysis and design grammars. Information Systems Journal 3(4), 217–237 (1993)
24. Wand, Y., Weber, R.: An Ontological Model of an Information System. IEEE Transactions on Software Engineering 16(11), 1282–1292 (1990)
25. Weston, N., Chitchyan, R., Rashid, A.: A framework for constructing semantically composable feature models from natural language requirements. In: The 13th International Software Product Line Conference (SPLC 2009), pp. 211–220 (2009)
26. WordNet, `http://wordnet.princeton.edu/`