Situation-Oriented Software Requirements Specification and Model Generation

1st Nimanthi L. Atukorala
Department of Computer Science
Iowa State University
Ames,USA
nimanthi@iastate.edu

2nd Carl K. Chang
Department of Computer Science
Iowa State University
Ames,USA
chang@iastate.edu

***Abstract*—In this paper, we present a semi-automated software requirements specification (SRS) and model generation methodology in order to formally represent the requirements elicited in human-centered fashion presented in our earlier study. The term situation is defined as a 3-tuple $< d, A, E >$ where d denotes human desire, A denotes the action vector, and E denotes the environment context vector. The probabilistic, timed situation-transition structure was derived using the observational data and was proposed to use as a source of new human-centered requirements elicitation. We illustrate the proposed methodology through some test cases with open access data sets and a comparison between existing SRS and the proposed SRS is given.**

***Index Terms*—Situation, Discrete-time, Markov chain.**

I. Introduction

A well-defined software requirements specification (SRS) and a model is the backbone of any successful software development process. Over the past few decades, many research studies have been conducted on providing more effective SRS format and, procedures including software tools and development environments where some are even commercially available. However, the main objective behind most of these approaches is limited to providing unambiguous specification formats to the users. In this paper, we propose a semi-automated software requirements specification and model generation methodology in order to formally represent the requirements elicited in human-centered fashion presented in our earlier study.

In an earlier study [1], we provide a human-centered definition for the term *situation* as a tuple $< d, A, E >$ where d represents the human desire, A represents a set of actions and E represents the environmental context. A probabilistic, timed transition structure between situations was then derived using the observational data. In [2] we proposed a methodology to elicit new software requirements by analyzing the existing dependency patterns in the derived. In this paper, we are extending our previous study to formally define the elicited requirements including possible assumptions, constraints, alternative requirements, and fit criteria. Finally, we propose a software model generation methodology based on the derived SRS in order to support the software designing and implementation phrase.

II. Probabilistic Timed Situation-Transition Structure (PTST Structure)

In 2009, [1] proposed a computational framework called Situ which includes a machine learning technique to infer human desire by considering human actions and environmental contexts. The term situation refers to a clear computational unit, which by this peculiar definition includes the human desire, behavioral and environmental contexts for a predefined time period. Traditionally, desire refers to a motivational mental state of a human subject [1]. Although a human desire is not visible, an earlier study, it is believed that the human behavior is related to their desire. Therefore, the desire can be inferred to some extent [3] through monitoring the user's activities and the environment where the developed system is operating within.

Definition: *A situation at time t, is a 3-tuple $\{d, A, E\}_t$ in which d is the predicted human desire (mental state), A is a set of human actions to achieve a goal which d corresponds to, and E is a set of environmental context values with respect to a subset of the context variables at time t.*

According to the above definition of the situation, for a particular person, if we know the desire, set of actions and environmental context values with time, it is possible to pair them as situation tuples and generate a sequence of situations with time. This sequence of situations may also include concurrent situations since a person might have multiple desires at a given time t, in which case the actions reflect multiple desires. One important observation is that some situations are leading to one or more future situations in the sequence with higher probability than others. We call this property as *situation transition*. Hence, it is possible to derive a situation-transition structure from the sequence of situations.

As a result of the possibility of concurrent situations, each node in the situation transition structure refers to a unique subset of situations. Our earlier study [2] describes a process of deriving situation-transition structure from observational data and existing causal relationships to elicit new requirements of the system.

In order to elevate this earlier study to next level, we proposed an enhanced version of situation-transition structure which we would like to call as *Probabilistic Timed Situation-*

Transition structure (PTST structure). The PTST structure is a domain-specific directed graph where each node represents a unique subset of situations and each directed edge represents a most likely transition from one node to another similar to the situation-transition given in [2]. However, each edge is now accompanied with a probability of the transition such that the sum of probabilities of transitions from a particular node to its successor nodes equals to one. Moreover, an edge also contains a time parameter that represents the average transition time[1].

III. Basic Definitions

We first formally define six basic entities that will be used in generating formal SRS of the system-to-be.

A. End-user

We use the term "end-user" to represent any human being who actually uses the system-to-be at the end. We assume that there are multiple end-users who use system-to-be simultaneously to perform different tasks. Each end-user has finite sets of frequent desires and actions. Moreover, end-users may divide into different categories based on one or more factors. These factors may range from different corporate titles corresponds to a company to age groups of family members. Hence, any end-user can be represented as,

$End-user: \ u =< ID, Category, Desires, Actions >$

Where,

ID – Unique identification

$Category$ – Specific group

$Desires$ – Finite set of desires

$Actions$ – Finite set of actions. Each action can be uniquely identify through the observation.

However, as mentioned in the previous study, the individual end-users may or may not be uniquely identified during the requirements situation-transition structure generation phrase and so do their categories. Hence, the value of two parameters ID and $Category$ can be $null$. In addition, the desires and actions included $Desires$ and $Actions$ sets may or may not be related to the system-to-be.

Note that we only consider the end-users as the human factors in this approach although there are other stakeholders involved in the requirements engineering process.

B. Object

A material thing that can be seen and touched by the end-users is referred to as an object. We assume that each object has a finite set of properties such as dimension, location ... etc. Objects may have a set of possible states either finite or infinite and each state of an object is measurable. Further, an object may change its' state to another state with time. Based on the cause of the change of states, all possible objects can be divided into two types as follows:

[1]Transition time of a transition from node A to node B is defined as the time interval between start of situation subset in node A to start of situation subset in node B.

Type1: Objects that change their states due to human action or human involvement.

Type2: Objects that change their states without any human involvement.

Therefore, any object can be represented as,

$Object: \ o =< P, ST, TYPE >$

Where,

P – Finite set of properties

ST - Set of possible states; discrete, finite/infinite, measurable

$TYPE$ – One of the two types of objects

C. Environmental factor:

Human behavior is sometimes affected by the environmental factors such as location, temperature, humidity,... etc. We assume that the value of any environmental factors related to human behavior is measurable with time. However, the possible values of an environmental factor can be either discrete or continues. Hence, the environmental factors can be divided into two types based on the possible values as follows:

Typed: Set of possible values are discrete. For example, location.

Typec: Set of possible values are continuous. For example, rainfall.

Therefore, the environmental factors are represented as,

$Environmental \ \ factor: \ \ e =< TYPE, VAL >$

Where,

$TYPE$ – One of the two types of environmental factors

VAL – Set of possible values (If the TYPE equals Typed, then this includes all the possible discrete values; If the $TYPE$ equals $Typec$, this includes a finite number of equal width ranges between upper and lower bound of the possible values. E.g. Fig. 1.)

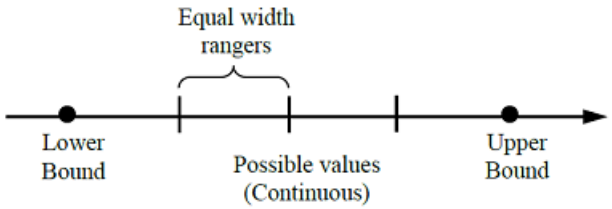

Fig. 1. Sample possible values in $Typec$ environmental factor

D. Data collecting components:

Data collecting components such as sensors are a subset of objects that used to record, end-user actions, states of other objects and the values of environmental factors in the domain with time. The type of these components (that is $Type1$ or $Type2$) depends on the type of recorded data. For example, data collecting components that record end-user actions and the states of $Type1$ objects belongs to $Type1$ whereas the data

collecting components that record states of $Type2$ objects and environmental factors belong to $Type2$.

In addition, a single data collecting component can be used for multi-purposes. For example, a data collection component aimed to record the states of objects that change their states due to human action ($Type1$) could also be used to record the existence of that particular user action.

E. Domain:

In domain engineering literature, the term *domain* is defined as an area of knowledge that scoped to maximize the satisfaction of the requirements of its stakeholders which includes a set of concepts and terminology understood by the practitioners in that area and the knowledge of how to build software systems (or parts of software systems) in that area.[4] The proposed approach represents the domain as,

$Domain: \ dom =< U, OBJ, ENV >$

Where,

U – Finite set of End-users; Each user in U has the form $< ID, Category, Desire, Actions >$

OBJ – Set of objects; Each object in OBJ has the form $< P, ST, TYPE >$

ENV – Set of environmental factors; Each environmental factor in ENV has the form $< TYPE, VAL >$

Note that this definition includes the assumption that only a finite number of end-users, user desires, actions, and environmental context values are possible within a domain. Multiple end-users in the domain can use system-to-be simultaneously to perform different tasks which imply simultaneous user desires and actions may occur. Similarly, simultaneous change of environmental factors is also possible.

F. Situation:

In [1] the term *Situation* is defined as a computational unit that encapsulates the human mental states, related behaviors and influenced environmental factors at a given time instant. Therefore, we represent any situation in a specific domain dom at a time instant t as,

$Situation: \ S =< d, A, E >_t \ \ in \ domain \ dom$

Where,

$d \in \bigcup_{u \in Set \ of \ end-users \ in \ dom} Desires \ of \ u$

$A \subseteq \bigcup_{u \in Set \ of \ end-users \ in \ dom} Actions \ of \ u$

$E = States \ of \ objects \ in \ OBJ \ in \ dom \ \cup \ Values \ of \ environmental \ factors \ in \ ENV \ in \ dom$

at time t.

Based on this definition, recording of situations in a particular domain means the recording of parameters d, A and E. However, as we assume in [1] that d can be inferred from the A and E, the recording of situations is narrowed down to the recording of A and E. According to the definitions of *domain* and *situation*, multiple situations can occur simultaneously at a given time instant.

IV. Situation-Oriented Software Requirements Specification

This section describes the methodology of deriving software requirements specification based on the situations and the properties extracted from the PTST. The proposed method encapsulates four hierarchical specifications: domain specification, situations specification, situation-causal specification, and software requirements specification; where each specification is derived using the definitions and properties given in the lower level specifications as given in Fig. 2.

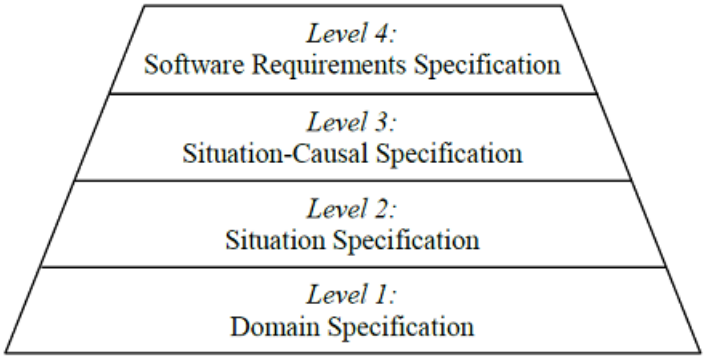

Fig. 2. Hierarchical specifications. Upper-level specifications are derived using lower-level specifications.

A. Level 1: Domain Specification

Domain specification is the baseline of the hierarchical specifications. As mentioned earlier, we represent any domain as,

$dom =< U, OBJ, ENV >$

Where,

U – Finite set of End-users

OBJ – Set of objects

ENV – Set of environmental factors

We use simple propositional logic to formally define a domain as a set of atomic propositions as follows:

1) Introduce individual constants for each of the following attributes in the domain.
   a) All possible end-users – $U1, U2, ...$ (Only if the individual users can be uniquely identified. Otherwise, all end-users are considered as one constant)
   b) All possible end-user desires – $D1, D2, ...$
   c) All possible end-user actions – $A1, A2, ...$
   d) Set of related objects including data collecting components – $O1, O2, ...$
   e) Two types of objects – $TYPE\_1, TYPE\_2$
   f) Set of environmental factors – $ENV1, ENV2, ...$
   g) Two types of environmental factors – $TYPE\_D, TYPE\_C$
   h) All possible states of objects – $ST1, ST2, ...$
   i) All possible states of environmental factors – $ENV1RANGE1, ENV2LOC1, ...$

2) Define set of predicates to represent the properties and relationships of the individual constants as follows.
   a) $CURRENT\_USER\alpha$ – α is the current user
   b) $CURRENT\_DESIRE\alpha$ – α is the current desire
   c) $STATE\alpha\beta$ – α is the state of object β
   d) $ACTION\_OCCURED\alpha$ – action α occurred
   e) $VALUE\_OF\alpha\beta$ – α is the value of environmental factor β

3) Define possible atomic propositions using the individual constants and the predicates as follows.
   a) $CURRENT\_USER(U1)$ – $U1$ is the current user
   b) $CURRENT\_DESIRE(D1)$ – $D1$ is the current desire
   c) $STATE(ST1, O1)$ – $ST1$ is the state of object $O1$
   d) $ACTION\_OCCURED(A1)$ – action $A1$ occurred
   e) $VALUE\_OF(ENV1RANGE1, ENV1)$ – $ENV1RANGE1$ is the value of environmental factor $ENV1$

Domain specification provides the definition of individual constants, predicated and the possible atomic proposition derived from them. We use the term *domain properties* to represent the set of all atomic propositions.

B. Level 2: Situation Specification

Each node in the $PTST$ structure represents a unique situation subset. Hence, we can define the union of situations represented in $PTST$ structure (SIT) as follows.

Let, NODES be the set of nodes in $PTST$ structure and SSi is the situation subset represented by node i in $NODES$. Then,

$SIT = \bigcup SSi \ \forall i \in NODES$

We present the situation specification as a mapping between each situation in SIT to a set of atomic propositions defined in domain specification. Since, a situation contains at most one desire, but any number of related actions and the contexts, this mapping is defined such that each situation maps to exactly one $CURRENT\_DESIRE$ but multiple $ACTION\_OCCURED$, $STATE$ and $VALUE\_OF$ format atomic propositions. Note that the Situation Specification acts as a labeling function (say SL) from SIT (set of situations in the $PTST$ structure) to the domain properties in domain specification (Fig. 3.).

C. Level 3: Situation-Causal Specification

Situation-causal specification formally specifies both observable causal relationships and the possible derived relationships among the situations in the $PTST$ structure. Note that, the $PTST$ structure along with situation specification can be considered as a discrete time labeled Markov chain

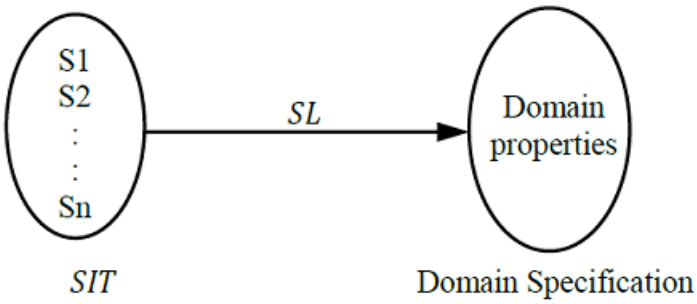

Fig. 3. Situation specification acts as a labeling function from set SIT to domain properties in the domain specification

(Probabilistic deterministic system) with $< S, T, L >$ format where the set of nodes and edges in $PTST$ structure represent the set of states S and the set of transition based probability function T respectively. A labeling function L for each state can be defined using the situation specification as follows.

Suppose state S is corresponds to node i in $PTST$ structure. Let SSi be the situation subset represented by node i. Then,

$L(S) = \bigcup SL(si) \ \forall \ situations \ si \ in \ SSi$

Hence, the causal relationships extracted from $PTST$ structure can be represented using the Probabilistic Real-Time Computational Tree Logic (PCTL) [5]. First, note that the following equivalences in the statements.

- $S_i \models Q$ or *"State S satisfies property Q"* $\iff$ $Q \in L(S)$ and the $PTST$ structure node corresponds to state S represents situation(s) where Q is an element of their domain properties.

- $S_i \models Q \wedge R$ or *"State S satisfies two properties Q and R"* $\iff$ $Q \in L(S)$ and $R \in L(S)$ and the $PTST$ structure node corresponds to state S represents some situations where either Q, R or both are elements of their domain properties.

In other words, Q and R may satisfy in a single situation or in different situations. In general,

$S_i \models Q1 \wedge Q2 \wedge ... \wedge Qn$ or *"State S satisfies properties Q1,Q2,... Qn "* $\iff$ $Q1, Q2, \ldots Qn \in L(S)$ and the $PTST$ structure node c corresponds to state S represents some situations where either $Q1, Q2, \ldots, Qn$ or subsets of $Qi's$ are elements of their domain properties.

1) ***Observable causal relationships****:* We consider the causal relationships directly extracted from the $PTST$ structure as the observable causal relationships. This subsection provides the definitions and the formal representation of the five observable causal relationships included in the situation-causal specification.

Let Q and R be two subsets of domain properties.

- **Direct cause:**
  Q direct causes R implies that almost surely the end of any state that satisfies Q but not R (say S_i) always coincides with the start of a state that satisfies R with at least probability p.

  $Q\ direct\ causes\ R \Rightarrow P_1((S_i \models Q \wedge \neg R) \rightarrow (S_i \models P_{\leq 1-p}(X\neg R))$
  Where,
  $P_{\leq 1-p}(X\neg R)$ implies $Pr_{Si}\{\pi \in Paths(S_i) \mid \pi \models X\neg R\}$
  $\pi \in Paths(S_i)$ such that $\pi[0] = S_i, \pi[1] = nodes\ proceed\ S_i$
  $\pi \models X\neg R$ implies $\pi[1] \models \neg R$ or $R \nsubseteq L(\pi[1])$

- **Leads to:**
  Q leads to R implies that almost surely paths start from any state that satisfies Q but not R (say S_i) is always contains a state that satisfies R within t_i time units with at least probability p.

  $Q\ leads\ to\ R \Rightarrow P_1((S_i \models Q \wedge \neg R) \rightarrow (S_i \models P_{\leq 1-p}(trueU^{\leq t_i}\neg R))$

- **Terminate:**
  Q terminates R implies that surely the end of any state that satisfies Q and R (say S_i) always start a state that satisfy not R within t_i time units with at least probability p.

  $Q\ terminate\ R \Rightarrow P_1((S_i \models Q \wedge R) \rightarrow (S_i \models P_{\leq 1-p}(trueU^{\leq t_i}R))$

- **Sustain:**
  Q sustains R implies that almost surely any state that satisfies Q and R (say S_i) is always followed by a state that satisfies R.

  $Q\ sustains\ R \Rightarrow P_1((S_i \models Q \wedge R) \rightarrow (S_i \models P_{\leq 1-p}(X\neg R))$

- **Prevent:**
  Q prevents R implies that almost surely any state that satisfies Q (say S_i) is not satisfies R and start of a state that satisfy R will not occur within t_i time units with at least probability p.

  $Q\ prevent\ R \Rightarrow P_1((S_i \models Q) \rightarrow (S_i \models \neg R) \wedge (S_i \models P_{\leq 1-p}(trueU^{\leq t_i}R)))$

*2) **Derived causal relationships**:* The observable causal relationships only based on the observed situation transition patterns in the domain within a period of time. However, it is possible to derive new causal relationships using a combination of those observable causal relationships. We believe that these relationships are useful in predicting the possible but yet unobservable situation transitions. The following derived causal relationships are inherited from Armstrong's axioms [6] in functional dependency in relational databases.

- **Causal relationships based on transitivity:**
  If Q direct cause Q' with probability p_1 and Q' direct cause R with probability p_2 then almost surely Q leads to R with probability $p_1 \times p_2$.

  If Q leads to Q' with probability p_1 and Q' leads to R with probability p_2 then almost surely Q leads to R with probability $p_1 \times p_2$.

  If Q terminates Q' within t_1 time units with probability p_1 and Q' terminates R within t_2 time units with probability p_2 then almost surely Q terminates R within $t_1 + t_2$ time units with probability $p_1 \times p_2$.

- **Causal relationships based on reflexivity:**
  For any relation among five observable causal relationships, if a relation holds between Q and R properties, then the same relation holds between Q and any subsets of R.

  E.g. Let Q direct causes R with probability p and let R' be any subset of R. Then, Q direct causes R' with at least probability p.

  Note that, the probability of the derived relation must be greater than or equal to the original relation.

- **Causal relationships based on augmentation:**
  For any relation among five observable causal relationships, if a relation holds between Q and R properties, then the same relation holds between $Q \cup K$and $R \cup K$, where K is a set of domain properties.

  E.g. Let Q direct causes R with probability p then, $Q \cup K$ direct causes $R \cup K$ with at most probability p.

  Note that, the probability of the derived relation must be less than or equal to the original relation.

D. Level 4: Software Requirements Specification (SRS)

Software requirements specification (SRS) contains a list of proposed new requirements of the system-to-be along with possible assumptions, constraints and alternative requirements derived using the causal relationships given in the situation-causal specification.

As the first step of defining SRS, requirements engineers must analyze the domain specification and categorized the domain properties into two groups as the domain properties that can and cannot be controlled by the system-to-be. Note that the new requirements are derived by only using the domain properties that controllable by the system-to-be.

TABLE I
SITUATION SPECIFICATION

Situation	Domain Properties
Situation A	d1,d2
Situation B	d2, d7
Situation C	d1,d3,d4, d8
Situation D	d5
Situation E	d4,d6,d7
Situation F	d1,d3, d8

However, the remaining domain properties can be used to identify the domain assumptions and constraints.

In order to derive the new requirements from controllable domain properties, we define a set of requirements construction terms based on one or more causal relationships in the situation-causal specification.

Let Q and R be two subsets of domain properties.

- **Achieve:**
  Achieve[R] [If Q hold in the current situation; Q leads to R then] sooner-or-later R
- **Maintain:**
  Maintain[R] [If Q hold in the current situation; Q sustains R then] always R
- **Avoid:**
  Avoid[R] [If Q hold in the current situation; Q prevent R then] always not R
- **Demote/Reduce/Minimize:**
  Demote/Reduce/Minimize[Q] [If Q hold in the current situation; Q terminate R then] sooner not R

V. GENERATING SOFTWARE MODEL USING SITUATION-TRANSITION STRUCTURE

According to our definition, the proposed SRS specifies the domain properties that can be controlled by the software-to-be. In this section, we describe a procedure to derive the software model using the situation-transition structure.

A. Identifying the components of the software-to-be

First, the components of the software-to-be must be identified by analyzing the domain properties in SRS. This can be done by grouping those domain properties by assuming that each group of domain properties can be completed, handled or controlled by a single component in the software-to-be.

For example, suppose for a particular domain there are eight domain properties $\{d1, d2, ..., d8\}$ defined in the domain specification and the mapping between these properties and the possible situations defined in the situation specification is as given in the TABLE I. Fig.4. shows the situation-transition structure for this domain.

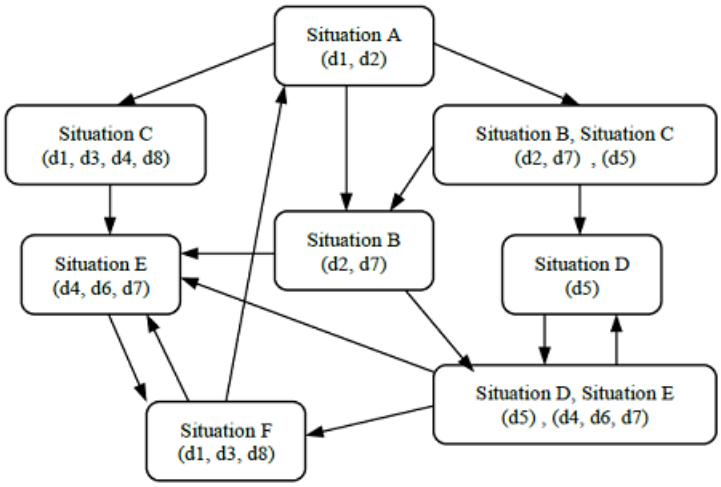

Fig. 4. Situation-transition structure of the domain

TABLE II
SOFTWARE-TO-BE COMPONENTS AND THE CONTROLLING DOMAIN PROPERTIES BY EACH COMPONENT

Sub-component of the software-to-be	Controlling Domain Properties
SC X	d1
SC Y	d3, d6, d8
SC Z	d5

Suppose the software-to-be can control the domain properties $\{d1, d3, d5, d6, d8\}$ and it can be done using three components as given in TABLE II.

B. Identifying the relations between components of the software-to-be

In order to identify the relations between components of the software-to-be, we introduce a new transition structure that embeds the software-to-be components into the situation-transition structure. The process of deriving the new transition structure may be required decomposition of situations in the original transition structure in order to replace them with the proposed software components. Fig. 5. shows derived new transition structure based on the original situation-transition structure in Fig. 4. and the defined components of the software-to-be in the previous section. Note that the resulted situation-transition structure now contains both possible human and software relations.

The software model can be obtained by filtering out the software components and the relations among them from the new transition structure. Fig. 6. shows the software model derived using the new transition structure in Fig. 5. This model can be modified further by deleting the redundant nodes and replacing the relations as given in Fig. 7.

Note that the derived software model can be used to further decompose the software components into smaller components as well as to identify new software components. For example,

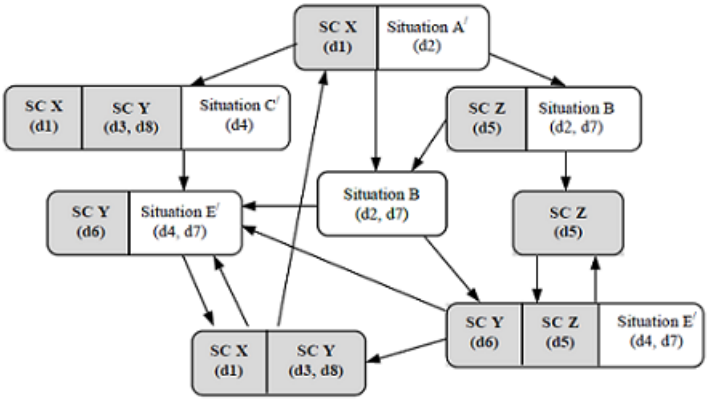

Fig. 5. Derived transition structure that includes both human and software relations

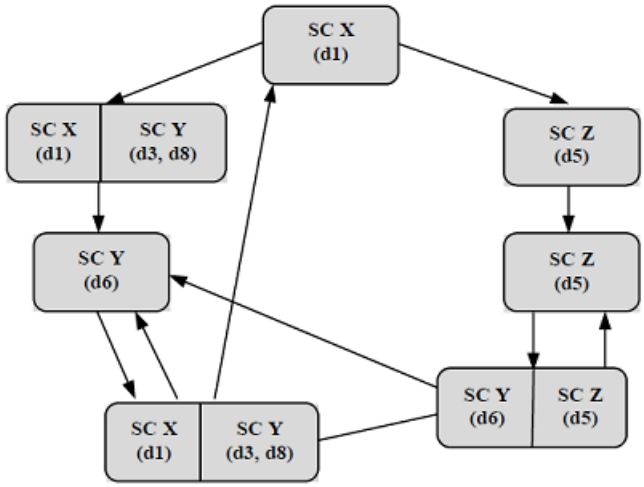

Fig. 6. Software model derived from the new transition structure in Fig. 5.

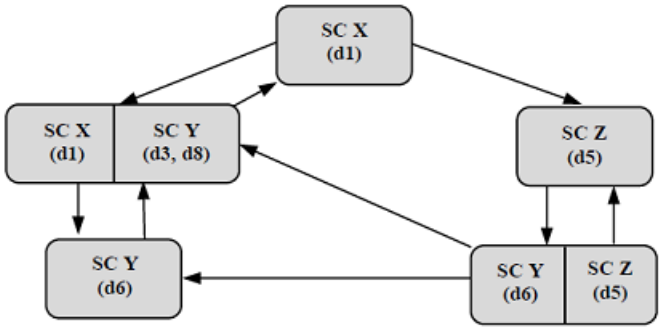

Fig. 7. Modified software model

we can decompose the software component $SC\ Y$ with domain properties $\{d3, d6, d8\}$ into two smaller components as $SC\ Y1$ with $\{d3, d8\}$ and $SC\ Y2$ with $\{d6\}$.

REFERENCES

[1] C.K. Chang, J. Hsin-yi, M. Hua and K. Oyama, "Situ: A Situation-Theoretic Approach to Context-Aware Service Evolution", Proc. IEEE Transactions on Services Computing, 2009, pp. 261 – 275.

[2] N.L. Atukorala, C.K. Chang and K. Oyama, "Situation-Oriented Requirements Elicitation", Proc. IEEE Computer Society International Conference on Computers, Software & Applications (COMPSAC) 2016, pp 233–238.

[3] J. Dong, C. K. Chang, and H. Yang, "Identifying Factors for Human Desire Inference in Smart Home Environments", Proc. Int. Conf. on Smart Homes and Health Telematics, 2013, pp. 230 – 237.

[4] K. Czarnecki and U. Eisenecker, "Generative Programming: Methods, Tools, and Applications", Boston: Addison Wesley, 2000.

[5] C. Baier and J.P. Katoen, "Principles of Model Checking", The MIT Press Cambridge, Massachusetts London, England, 2008.

[6] W. Armstrong, "Dependency Structures of Data Base Relationships", TIFIP Congress, 1974, pp. 580-583.