2018

Situation-Transition Structure and Its Applications in Software System Development

Nimanthi L. Atukorala
Iowa State University, nimanthi@iastate.edu

Carl K. Chang
Iowa State University, chang@iastate.edu

Katsunori Oyama
Nihon University

Situation-Transition Structure and Its Applications in Software System Development

Abstract
Observing, analyzing and understanding human factors is becoming a major concern in software development process in order to gain higher customer satisfaction. In this paper, we present a semi-automated methodology to generate the situation-transition structure which can be used to analyze the human behavior patterns in a specific domain. The term situation is defined as a 3-tuple < d; A;E > where d denotes human desire (mental state), A denotes the human actions vector, and E denotes the surrounding environment context vector. The situation-transition structure is a directed weighted graph where each node represents a unique situation or set of concurrent situations and an edge represents the transition from one situation to another. Data mining and machine learning techniques are used to generate situation-transition structure from raw observational data. We illustrate the proposed methodology through some case studies with open access datasets. The applications and advantages of situation-transition structure in software development are then asserted.

Keywords
software development, situation, situation-transition structure, directed weighted graph, data mining, machine learning

Disciplines
Computer Sciences | Software Engineering

Situation-Transition Structure and Its Applications in Software System Development

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

3rd Katsunori Oyama
Department of Computer Science
Nihon University
Koriyama, Japan
oyama@cs.ce.nihon-u.ac.jp

***Abstract*—Observing, analyzing and understanding human factors is becoming a major concern in software development process in order to gain higher customer satisfaction. In this paper, we present a semi-automated methodology to generate the situation-transition structure which can be used to analyze the human behavior patterns in a specific domain. The term situation is defined as a 3-tuple $< d, A, E >$ where d denotes human desire (mental state), A denotes the human actions vector, and E denotes the surrounding environment context vector. The situation-transition structure is a directed weighted graph where each node represents a unique situation or set of concurrent situations and an edge represents the transition from one situation to another. Data mining and machine learning techniques are used to generate situation-transition structure from raw observational data. We illustrate the proposed methodology through some case studies with open access datasets. The applications and advantages of situation-transition structure in software development are then asserted.**

***Index Terms*—software development, situation, situation-transition structure, directed weighted graph, data mining, machine learning.**

I. Introduction

The success of any software system relies on their end-users' satisfaction. Hence, modern software development companies are putting more effort to include novel features and technologies in their products. Often those novel features include personalization, high efficiency, high security, and better usability so that the software system can provide a unique experience to each and every end-user. However, in practice, there are situations where some of those highly personalized software systems are still unable to reach the expected customer satisfaction. We believe that the lack of computational methodologies to track the end-users' genuine individual factors including their interests, beliefs and lifestyle plays a major role in such failures. Although the effects of individual human factors to the success of a software product are extensively studied in Social Sciences research, providing a computational solution to sufficiently embed those factors to software development process still remain a key challenge. Moreover, the complex and changing nature of humans makes it more difficult to find such a solution.

In the prevailing software development process, interviewing the end-users is the most commonly used methodology to identify their needs. However, finding genuine individual factors of end-users through interviews could be time-consuming, ineffective, and sometimes impossible. On the other hand, observing human behavior is another way to collect information on individual human factors. With the rise of the smart devices and smart environments, observing each individual end-user has become more precise, effective and inexpensive.

When considering human behavior, it is important to note that those behaviors are often triggered by their mental states. Hence, the effects of the behavior may differ according to the mental state. For example, the effect of a particular behavior with good desire may not be the same as the effect of the same behavior with ill desire. Moreover, human behavior also depends on the surrounding environment. An unusual behavior in one environment may be an expected behavior in another environment. These facts show that in order to effectively monitor, analyze and detect changes in human behavior, the information on the human mental state and the surrounding environment is important.

The main objective of this paper is to introduce a computational methodology to identify human behavioral patterns using raw observational data which can be used as a source to understand and embed the individual human factors to software development process. We believe that the individual human factors are directly connected to their goals and desires; or in general, their motivational mental state [1]. In order to configure the human mental state into a computational model, the proposed method borrows the concept of situations presented in an earlier study [1]. The term situation refers to a clear computational unit, which by this peculiar definition includes the human mental state, behavioral and environmental contexts for a predefined time period. Although a human mental state is not visible, in an earlier study, it is believed that the mental state can be predicted to some extent [2] through monitoring the human activities and the environment. Thus, a sequence of situations can be obtained for each individual as in Fig.1.

Identification of precise boundaries of actual situations in such a sequence is challenging, and no standard methodology has been defined. In our method, we use concepts in frequent patterns mining [3] to discretize the time as situation boundaries. The significance of using frequent patterns

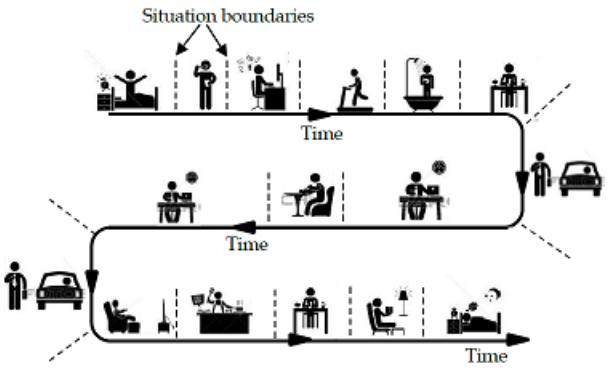


Fig. 1. A sample sequence of situations based on a daily routine of an individual.

mining over the other possible discretizing techniques such as entropy-based discretization [4] is its ability to adjust in different domains and applications. This discretization of time will eventually result in a sequence of situations with time. Next, the transition structure of the sequence of situations is generated by applying a computational model based on the Chow-Liu Bayesian network structure learning algorithm [5]. Although finding the precise Bayesian network structure from data is NP-hard [6], the Chow-Liu algorithm provides a good approximation using joint probability distribution. Calculation of joint probability distribution between situations in a sequence is more likely than calculating required factors in other Bayesian structure learning algorithms such as independence between two situations in the PC algorithm [6].

The rest of this paper is organized as follows: Section 2 outlines the proposed method to generate human situation-transition structure in detail. In Section 3 a summary of real-life case studies is employed for illustration. Section 4 includes a discussion on some applications and advantages of situation-transition structure in different stages of the software development process and finally, Section 5 concludes the paper.

II. Situation-Transition Structure Generating Process

In this section, we describe the situation-transition structure generating procedure in detailed.

A. Situ Framework

Situ defined in [1] is a computational framework which describes the process of inferring human desires through relevant human actions and the environmental contexts. The term "situation" defined in situ framework encapsulates these three factors; that is a human mental state, actions, and environmental context into a single computational unit. Our proposed methodology uses this definition of the situation in order to comprise the human mental state to the situation-transition structure.

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.*

Based on this definition of the situation, it is possible to pair time-stamp records of desires, set of actions and environmental context values of a particular person into situation tuples which leads to a sequence of situations with time. Moreover, it is possible that a person may have multiple desires at a given time t and the actions perform during that time reflect one or more of those multiple desires. In other words, a sequence of situations of a person may include time periods where multiple situations occur simultaneously. Fig.2. shows some possible concurrent situations.

Fig. 2. Concurrent situations: (a) Eating while driving. (b) Watching TV while exercising.

One significant observed property in situation sequence of any person is that some situations are more likely to transfer to a specific subset of future situations than others. We define this property as *situation transition*. Note that these situation transitions are domain dependent. Hence the term situation-transition structure in this study refers to a domain-specific directed weighted graph generated using all possible situation transitions in a particular domain.

B. Situation-transition structure generating process

Fig.3. represents the overall process of the proposed method. The process starts with (1) identifying relevant human subjects, the domain of interest and collecting initial raw data by observing the regular activities of those human subjects in the domain. Next, (2) the collected raw data are pre-processed in order to extract the behavioral and environmental context with time. (3) The definition of the situation is then employed to encapsulate the data into more computationally rich information units which can be used to (4) derive a sequence of situations with time. Finally, (5) the situation-transition structure is generated by applying machine learning based computational model to the derived situation sequence. Note that the generated situation-transition structure can be used to extract information on human interest, believes, lifestyle and even existing environmental or external constraints.

1) Choosing the significant human subjects and the domain: Selecting the relevant human subjects and the domain is the first stage of generating the situation-transition structure. The selection of human subjects depends on the primary goal of

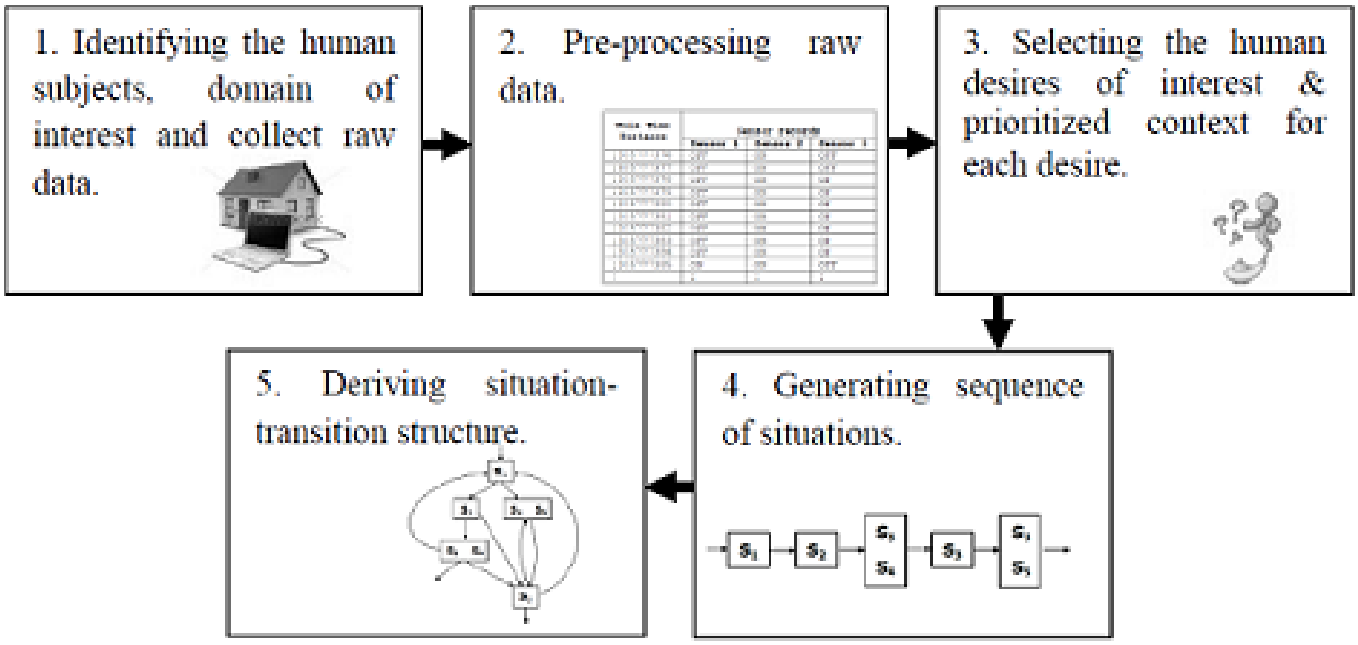


Fig. 3. The overall process of proposed Situation-transition structure generating.

the situation-transition structure generation. The term domain implies the environment where the situations of interest are most likely to occur. Selection of domain also depends on the primary goal of the situation-transition structure generation as well as the particular human subjects.

For example, in software engineering the human subjects are called stakeholders that refer to any individuals or groups or organization that may affect, affected by the developed software. These stakeholders can be divided into two main categories as development team members and the end-users. Although the individual human factors of development team directly affect to the effectiveness of the software development process, the final decision on successfulness depends on the end-users' satisfaction. Hence, focusing on end-users as human subjects is one option in generating the situation-transition structure for software development applications.

Note that the selected set of human subjects and the boundaries of the domain can be updated during the process but may increase the time duration of the entire process and resource consumption.

2) Collecting raw data: Next, raw data must be collected through observation of both the regular activities of the human subjects and the relevant environmental factors in the domain. The indirect observation can be performed through data collecting component such as sensors. Each data collecting component is uniquely labeled as $ComponentType : Location : ID$. Data collected from these components must be recorded with time.

Appendix A gives details on different approaches to observe the activities of the human subjects and the environmental factors as given in Fig. 4.

The observation is conducted without any kind of interaction with the participants, assuming that prior agreement has been reached and the participants know that they are being observed. Further, there are no particular activities that are planned to observe in advance. All activities were given equal priority. Therefore, according to Robson [9], the observation in the proposed method can be classified as an unobtrusive and unstructured approach. We believe that these properties increase the flexibility and, thus the effectiveness of the proposed method. Moreover, the duration of data collection can be decided by domain expert based on the human subjects and the domain of interest.

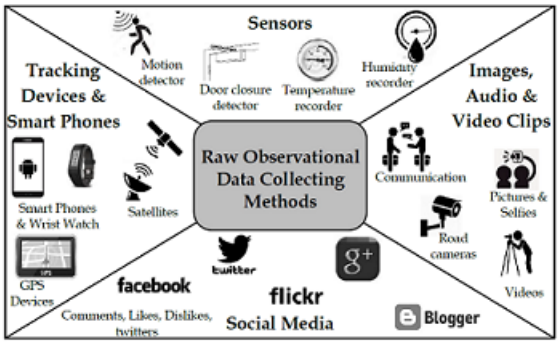


Fig. 4. Various methods of collecting raw observational data. More information: Appendix A.

3) Identifying of Behavioral and Environmental Context Components: Raw data collecting components can be divided into two categories as Behavioral Context Components (BCC) and Environmental Context Components (ECC). Components that provide data about activities such as motion belong to BCC, whereas components that provide data about environ-

mental factors such as location, temperature belong to ECC.

We assume that the states of ECC can be in two forms as discrete states and continuous states. For example, the state of ECC that records location can be one of the possible locations defined using the coordinate values, however, the state of ECC that records temperature can be laid in the range -30.0 °F to 120.0 °F. Note that in either case, ECC does not imply any activity been occurred or not.

Conversely, we assume that all the BCC have set of known discrete states based on whether it "sensed a signal" or not, and can be recorded with time whenever their states changed from one to another. The states remain the same during the time intervals between two twists of states. Note that the time intervals between two twists are not regular. For example, the state of a BCC use for motion detection can be either ON or OFF and the twist from ON state to OFF state at time t1 and OFF state to ON state at time t2 can be recorded along with time t1 and t2. Then, the BCC state remains OFF from t1 to t2 period of time. Moreover, we assume that all possible states of BCC can be categorized as "positive states" and "negative states" (Fig.5.).

Definition: *Positive states of a BCC are the set of states to represent that BCC "sensed a signal" or the states that indicate "activity had occurred". Negative states of a BCC are the set of states to represent that BCC "did not sense a signal" or the states that indicate "activity had not occurred".*

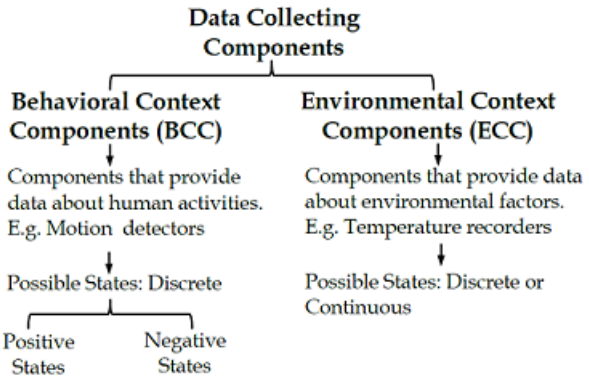


Fig. 5. Two categories of raw data collecting components and their possible states.

Analysts or domain experts can select the positive and negative states for BCC based on their type and the functionality. However, differentiating between positive and negative states could be difficult in cases where data collecting components engaged in multiple activities. The selection of positive and negative BCC can be considered in two cases.

Let S be the set of all possible states of a single behavioral context component.

- **Case 1:** If all states in S correspond to a single activity, then let $P \subseteq S$ represents the particular activity had occurred and the complement of P (say P') represents the activity had not occurred. Then, select P as positive states set and P' as negative states set. If P' is empty, introduce a fictional state as $NULL$ for the negative state. For example, if a motion detection sensor has two states, $S = \{ON, OFF\}$, and the ON state represents a "movement had occurred in a particular area" whereas OFF state represents "no movement occurred in a particular area", then, the subset $P = \{ON\}$ becomes the positive state, while $P' = \{OFF\}$ becomes the negative state. Another example is if the two states $S = \{OPEN, CLOSE\}$ of a door-closure sensor represent the same activity, then $P = S = \{OPEN, CLOSE\}$ becomes the positive states and $P' = \{NULL\}$ becomes the negative state which is fictional.
- **Case 2:** If states in S correspond to different activities, then consider that component as a group of separate components where each component i in the group has a subset Pi of S to denote that the same activity ai had occurred. All the remaining states in S, i.e. the complement of Pi (say Pi'), denote that ai had not occurred. Again, select Pi as positive states set and Pi' as negative states set of component i. If Pi' is empty, introduce a fictional state. For example, if $S = \{ON, OFF\}$ for a switch-state detection sensor represents three different activities X, Y and Z, such that ON state represents either X or Z had occurred and OFF state represents either Y or Z had occurred, then that sensor can be regarded as a group of three sensors with positive states set $Px = \{ON\}$, $Py = \{OFF\}$, $Pz = \{ON, OFF\}$, and negative states set $Px' = \{OFF\}$, $Py' = \{ON\}$, $Pz' = \{NULL\}$, respectively.

Note that, the selection of positive and negative states will introduce a new set of BCC where each component state corresponds to a single activity. In the rest of the paper, the term BCC refers to this new set of BCC.

4) Pre-processing raw data: During the pre-processing of raw data, we discretize the time into small equal-width intervals, and the state of each BCC is recorded at the end of each time interval. All BCC that do not have positive states at these time instances will be recorded as in their negative states. The analysts can determine the width of the time interval. Reliability of the process will increase for smaller time intervals.

In addition, states of each ECC at these instances are recorded separately. If a state of ECC varies within the time interval, then the average value can be computed and recorded.

The difference between the recorded times of the previous record and the current record is called the time interval of the current record. Our methodology assumes that the BCC states are constant within this time interval. Amount of information lost due to this assumption can be reduced by selecting smaller time intervals.

5) Selecting the human desires of interest and prioritization of BCC states for each desire: Next step in the proposed method is to select a sample of desires that have influences on the system-to-be. This step needs to be accomplished by analysts with prior knowledge of human subjects, domains of interest, and the system-to-be. Quick interviews with partici-

pants can be helpful during this process. The granularity of the desires depends on human subjects, the domain of interest and the system-to-be. After selecting the human desires of interest, the analysts need to provide a priority BCC states set for each desire.

Definition: *Priority BCC states set of a desire is a subset of BCC states which has higher implication with respect to the particular desire.*

Our definition of priority states set of a desire only consists of positive states since they provide a clear indication of the occurrence of activities. However, this definition can be altered if the negative states are also significant with respect to some desires. Analysts can choose this set for each desire using their background knowledge. Although the involvement of analysts is essential throughout this step, semi-automated tools can be developed to support them to make better decisions.

6) Generating a sequence of situations: As given earlier, the definition of a situation at time t gives a 3-tuple d, A, E. This section describes the method to generate a sequence of situations by using pre-processed BCC records and preprocessed ECC records to identify action set A and environmental context set E for each situation. Then the corresponding human desire d can be inferred. Frequent patterns mining [3] is one of the common tasks in data mining, which aims at identifying a subsequence that appears in a data set with a frequency that is greater than or equals to some specified threshold. In our method, we define frequency pattern as follows.

Definition: *A frequency pattern (p) is a subset of positive BCC states that occurs frequently with the time. The pattern length is the number of BCC states in that pattern.*

According to the definition, a frequency pattern only contains positive BCC states by eliminating both negative BCC states and ECC states. This will allow us to focus on activities been performed rather than activities not been performed. Also, it helps to reduce the pattern length since at a given instant there is a fewer number of positive BCC states than the negative BCC states. However, this definition can be altered based on the significance of negative BCC states. Suppose that pi represents the ith pattern. Then, there exists a set of frequency patterns $FP = \{p1, p2, \ldots, pn\}$ in preprocessed BCC records where the number of patterns depends on the number of BCC and how related their positive states could be. It is possible that two frequency patterns share the same set of states and therefore, some longer patterns may include one or more shorter patterns.

As our first step of identifying actions set A, we obtain the set of frequency patterns FP that appears in the pre-processed BCC records. We use the FP-Growth algorithm [3] for this task due to its high performance while any frequency pattern mining algorithm will be appropriate. This step can be fully automated.

Once the set of frequency patterns FP is obtained, the next step is to identify the original preprocessed BCC records where each of these patterns exists. Proposed algorithm to match the original preprocessed BCC records with the patterns in FP is given in Algorithm 1. It takes the set of frequency patterns FP and preprocessed BCC records DS as the input.

```
Data: FP : Set of frequency patterns, DS : Processed
      BCC records
Result: Processed BCC records labeled with patterns
        and final set of frequency patterns used
Read the Input datasets
sort_FP : Sorted FP on descending order of pattern
 lengths
temp_DS : DS    /* Initialize to original
 DS */
temp_FP : sort_FP          /* Initialize to
 original sort_FP */
while temp_DS is not empty do
    for each record r in temp_DS do
        for each pattern pi in temp_FP do
            if r contains pi then
                Label r with pi
                Remove r from temp_DS
            end
        end
    end
    temp_FP : Get the new set of frequency patterns by
     taking remaining records in temp_DS as the input
    sort_FP : Set of (sort_FP ∪ temp_FP) sorted on
     descending order of pattern lengths
end
sort_FP : Final set of frequency patterns used to label
 records in DS sorted on descending order of pattern
 lengths
```

Algorithm 1: Pattern selection algorithm for preprocessed BCC records

The labeled preprocessed BCC records for each pattern can be further processed by combining the consecutive records with same pattern labels together to form a single record. We call the final set of labeled records as **"concrete pattern records"** and their time intervals as **"concrete pattern intervals"** since each of these labeled records corresponds to a particular frequency pattern. Note that the concrete pattern intervals are discrete. In the end, this will result in a sequence of concrete pattern records and a corresponding sequence of concrete pattern intervals. In the proposed method, we assume that the sequence of concrete pattern intervals introduces the time boundaries of situations, and the set of unique BCC states (both positive and negative) in concrete pattern records forms the set of actions occurred during that time interval. Moreover, we assume that the ECC states form the set of environmental contexts. Note that, the state of a particular ECC can be varied during a concrete pattern interval and therefore, each ECC state within a concrete pattern interval is recorded as the 3-tuple: $< minimum, maximum, mean >$.

At a given time t, a person or group of people may have multiple desires. Therefore, the BCC states in a concrete pattern record may correspond to multiple desires. However, the positive BCC states in a concrete pattern record are the same as its labeled frequency pattern and the priority BCC

states set of a desire only includes positive BCC states by definition. In other words, it is possible to introduce frequency patterns and desires mapping algorithm as given in Algorithm 2. This algorithm takes the final set of frequency patterns FP used to label preprocessed BCC records, the set of desires of interest DI, the priority BCC states set of each desire DP and a user-defined threshold value T as the input.

The accuracy of this mapping can be improved by updating the priority BCC states set of existing desires or introducing new desires and corresponding priority BCC states sets by requirements analysts. The above process will generate a set of actions, the set of environmental contexts and possible desires for each concrete pattern intervals.

```
Data: FP : Final set of frequency patterns used to label
      preprocessed BCC records, DI : Set of human
      desires of interest, DP : Priority BCC states sets
      for each desire, T : User define threshold value
Result: Set of possible desires for each pattern in FP
Read the Input datasets
for each pattern pi in FP do
    pi_bcc : Set of states in pi
    pi_desireList : List of possible desires for pattern
      pi, initialize to empty list
    for each desire dj in DI do
        dj_priorityBCCStates : Set of Priority BCC
          states of desire dj
        pi_dj_jaccard : Jaccard similarity between
          pi_bcc and dj_priorityBCCStates
        if pi_dj_jaccard ≥ T then
            Add dj to pi_desireList
        end
    end
end
```

Algorithm 2: Frequency patterns and Desires mapping algorithm

According to the definition given in [1], a situation can contain at most one desire. Hence, two cases need to be considered when deriving sequences of situations.

- **Case 1:** If a concrete pattern interval mapped into only one desire, then that particular desire along with the set of actions and environmental contexts within the concrete pattern interval can be considered as one situation.
- **Case 1:** If a concrete pattern interval is mapped to multiple desires, it implies multiple situations had occurred simultaneously within the interval. Then, the set of actions during that time can be separated into subsets belonging to each desire based on the existence of them in the priority BCC states of desires. Note that, some actions can be common to multiple desires whereas some others do not belong to any desire. In the latter case, we can either ignore those actions or consider them as common to all the desires. A number of actions not belonging to any desire can be reduced by either updating the priority BCC states of desires or introducing new desires and corresponding priority BCC states sets. Note that the set of environmental contexts does not depend on the desire. Therefore, each tuple of desire d, a set of actions A corresponding to d, set of environmental contexts E within the concrete pattern interval forms multiple simultaneous situations within the concrete pattern interval.

Note that the resulting sequences of situations from the above process may contain too many similar situations with same desire d and same or slightly different sets A and E. Consideration of these similar situations as unique situations during the situation-transition structure deriving process will include redundant causal relationships and increase both time and space consumption. As a solution, we introduce a classification of situations into discrete groups. i.e., we further process the sequences of situations such that the situations with the same desire d and same or slightly different sets A and E are classified into one group. This will introduce discrete sets of situations and each set is given a unique name as $S_{d1}, S_{d2} \ldots S_{dn}$. For a particular set S_{di}, consider the common desire as d, the union of actions sets belongs to situations in S_{di} as A and the overall minimum, maximum, mean of environmental contexts belong to situations in S_{di} as E. Subsequently, the situations in the original sequence are renamed by their corresponding set names, which will result in a processed sequence of situations.

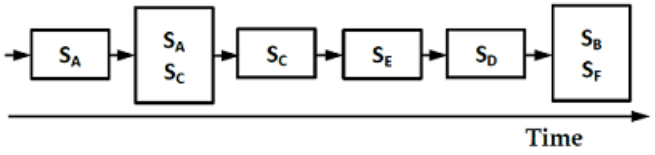


Fig. 6. A sample of a preprocessed sequence of situations.

Fig.6. shows a sample of a preprocessed sequence of situations. Such a sequence of situations can be used to define the situation-concurrency set as follows.

Definition: *The situation-concurrency set of a situation sequence is the union of individual situations and groups of simultaneously occurred situations in that sequence of situations.*

For example, the situation-concurrency set of the situation sequence given in Fig.6. includes three elements as $\{S_A, S_C, S_D, S_F, \{S_A, S_C\}, \{S_B, S_F\}\}$.

7) Deriving situation-transition structure: This section describes the process of identifying a causal relationship within the derived sequence of situations. Here we use the concept of Bayesian networks [6]; a well-known machine learning technique to infer the causal relationships between different factors based on observational data. Our methodology uses an algorithm called the Chow-Liu algorithm [5] as the baseline. In order to preserve information on sequential ordering of input data and to allow the presence of cycles, some modifications are made to the original algorithm. Algorithm 3. describes the modified Chow-Liu algorithm.

The algorithm takes a processed sequence of situations $SITU\_SEQ$, the situation-concurrency set $SITU\_CON$ and a user-defined threshold value T as the input. The logical

matrix is generated in order to calculate the mutual information between each pair of elements in the $SITU\_CON$. The use of the threshold value to decide the existence of edges allows the presence of cycles. Note that, the calculations of probabilities required a single $SITU\_SEQ$ travel which has linear time complexity.

A sample of the situation-transition structure is given in Fig.7. Note that this algorithm only considers the direct transitions from one situation to another in situation sequence when generating the transition structure. In another word, some it is possible to lose some information about the indirect transitions between situations in the sequence. Appendix B proposes a sliding window based approach to minimize this information lost.

8) Extension of situation-transition structure to probabilistic, timed situation-transition structure: Note that the derived situation-transition structure in the previous section can be used to filter out the most common transitions. This structure can be further enriched with transition probabilities and time took for the transition by analyzing the resulted transitions relations in the original situation sequence. We call the final resulting structure as probabilistic, timed situation-transition structure.

III. Case Studies

We have applied our proposed situation-transition structure generation methodology for real-time data sets in different application areas in order to demonstrate its applicability in real-world system development.

- **Case Study 1: Smart Home energy management system** Smart* dataset [10] contains sensor data that were collected in a two-story, 1700 square foot home with three full-time occupants during May 01, 2012 – July 31, 2012 time period (306772 records). The data collecting components include motion sensors, door-closure sensors, switch-state detection sensors, temperature sensors, heat index sensor, humidity sensor, wind-chill sensor, wind-speed sensor, and rainfall measurements. We identified twelve possible desires such as prepare meal, wash dishes, eat, sleep, and laundry.
- **Case Study 2: Cooperative Research Environment (CoRE)** CoRE website is aimed for sharing published and unpublished internal research papers, comments and ideas in a research environment. The dataset [11] contains the records on user operations on CoRE website interface by 65 users between September 16, 2014 – October 21, 2014 time period (10062 records). The data collecting components include button and link click monitors, menu option selection monitors, current and next web-pages. We identified eighteen end-user desires such as login, view user profile, upload a paper, download a paper, and submit a comment.

IV. Application of human situation-transition structure in software development

As mentioned earlier, understanding the end-user behavior has many applications in the software development process. Such a structure can be used for real-time software development and update. In this section, we describe some of these applications in detail.

- **New software requirements elicitation using situation-transition structure**
  In [7] we explain a systematic approach to elicit new software requirements by analyzing the causal relationships in situation-transition structure. This process can be used to identify requirements to develop completely new software or to update any existing software. Note that the requirements gathered from this process are domain-specific; meaning the set of requirements for the software will be differed according to the environment is deployed. Moreover, the proposed requirements elicitation approach can be used to identify individualized requirements[1]; meaning that it is possible to identify set of requirements for each individual user that depends on their personal believes, goals and desires. We believe that the use of domain-specific individualized requirements in developing software may lead to higher customer satisfaction than using a general set of requirements.
- **Software designing and implementation**
  Situation-transition structure can be used to identify the user preferences, existing limitations, and threats as well as the disagreements of the requirements. It is also useful to identify the required software components and to introduce the possible relationships among them. In [8], Ming et al. introduced a functional programming language that can be used to implement a software design based on the situation.
- **Software testing**
  The information gathered from the situation-transition structure can be used to derive the domain-specific, individualized test cases in order to verify and validate any software or prototype. Note that once the software or the prototype is developed, the structure generating procedure can be repeated to generate a new situation-transition structure that includes the software itself. This new structure can be used to identify the unexpected behaviors and failures of the current version. Moreover, a general set of test cases can be derived by comparing situation-transition structures generated for the same software in similar domains.
- **Software maintenance and evolution** Regular generation of situation-transition structure for the same software or comparison between different existences of the software can be used to identify any faults in the software.

Although our proposed computational model can be used in any software development process, we believe that it is

[1]Individualized requirements are not same as personalized requirements.

Data: SITU_SEQ : Sequence of situations with time, SITU_CON : Situation-concurrency set, PERCENTAGE : Percentage of Mutual Information to be selected as Threshold
Result: Situation transition structure
Read the Input datasets
num_Transitions: Number of situation transitions in SITU_SEQ. Initialize to zero.
SITU_AVAILABILITY = Mapping of each elements in SITU_CON with an integer array of two elements. All elements initialize to zero.
SITU_PAIR_RELATION = Mapping of each possible pair of elements in SITU_CON with an integer array of four elements. All elements initialize to zero.
num_nodes = Cardinality of SITU_CON
MI = Two dimensional floating-point array with size num_nodes X num_nodes to store Mutual Information
maxMI = Maximum value of Mutual Information
minMI = Minimum value of Mutual Information
T = Threshold value

/* Defining mappings */

for *each x in SITU_CON* **do**
    a = Integer array with two elements. All elements initialize to zero.
        /* a[0]=Number of occurrences where x does not appear in a pair of nodes in SITU_SEQ */
    /* a[1]=Number of occurrences where x appears in a pair of nodes in SITU_SEQ */
    Map x with a in SITU_AVAILABILITY
    for *each y in SITU_CON* **do**
        b = Integer array with four elements. All elements initialize to zero.
            /* b[0]=Number of occurrences where x,y is not the current $< parent, child >$ pair of situations in SITU_SEQ */
        /* b[1]=Number of occurrences where x is not the current parent but y is the current child situations in SITU_SEQ */
        /* b[2]=Number of occurrences where x is the current parent but y is not the current child situations in SITU_SEQ */
            /* b[3]=Number of occurrences where x,y is the current <parent,child> pair of situations in SITU_SEQ */
        Map $< x, y >$ pair with b in SITU_PAIR_RELATION
    end
end
current_parent = First situation in SITU_SEQ
for *each current_child in SITU_SEQ starting from second situation* **do**
    b = Integer array mapped to $< current\_parent, current\_child >$ pair in SITU_PAIR_RELATION
    b[3] = b[3] + 1
    for *each p in SITU_CON* **do**
    end
    if *$p \neq$ current_parent and $p \neq$ current_child* **then**
    end
    otherPB = Integer array mapped to $< p, current\_child >$ pair in SITU_PAIR_RELATION
    otherPB[1] = otherPB[1] + 1
    for *each c in SITU_CON* **do**
        if *$c \neq$ current_parent and $p \neq$ current_child* **then**
            otherCB = Integer array mapped to $< current\_parent, c >$ pair in SITU_PAIR_RELATION
            otherCB[2] = otherCB[2] + 1
        end
    end
    num_Transitions = num_Transitions + 1
            /* Assign current_child as the current_parent for the next iteration */
    current_parent = current_child
end

/* (Continue on next page) */

Let a, b be an integer array.
for *each x in SITU_CON* **do**
  for *each y in SITU_CON* **do**
    if $x \neq y$ **then**
      b = Integer array mapped to $< x, y >$ pair in SITU_PAIR_RELATION

$$b[0] = num\_Transitions - \sum_{i=1}^{3} b[i]$$

    end
  end
  a = Integer array mapped to x in SITU_AVAILABILITY
  a[0] = b[0] + b[1]
  a[1] = b[2] + b[3]
end

```
/* Generate situation dependency structure */
```

for *each x in SITU_CON* **do**
  Create a node in the graph.
end
for *each x in SITU_CON* **do**
  ax = Integer array mapped to x in SITU_AVAILABILITY
  x0 = ax[0]
  x1 = ax[1]
  for *each y in SITU_CON* **do**
    ay = Integer array mapped to y in SITU_AVAILABILITY
    y0 = ay[0]
    y1 = ay[1]
    if $x \neq y$ **then**
      b = Integer array mapped to $< x, y >$ pair in SITU_PAIR_RELATION
      xy00 = $\frac{b[0]}{num\_Transitions}$
      xy01 = $\frac{b[1]}{num\_Transitions}$
      xy10 = $\frac{b[2]}{num\_Transitions}$
      xy11 = $\frac{b[3]}{num\_Transitions}$

```
/* Calculate mutual information of <x,y> */
```

      if *x0>0 and x1>0 and y0>0 and y1>0 and xy00>0 and xy01>0 and xy10>0 and xy11>0* **then**

$$MI[x][y] = \sum_{i,i=0,1} xyij \times log(\frac{xyij}{xi \times yj})$$

        Update maxMI and minMI
      end
    end
  end
end
$T = (maxMI - minMI)XPERCENTAGE$
for *each x in SITU_CON* **do**
  for *each y in SITU_CON* **do**
    if *MI[x][y]* $\geq T$ **then**
      Create a directed edge from node x to y
    end
  end
end

Algorithm 3: Situation-transition structure generating algorithm

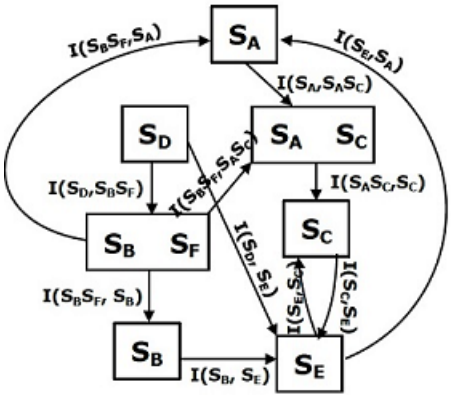


Fig. 7. A sample of the situation-transition structure

more effective in development of socio-technical systems and safety-critical systems.

In the era of smart environments and Internet of things, most of the software systems are part of larger socio-technical systems [12] which include not only the technical components, but also social ones such as humans and organizations. [13] Hence, understanding the end-user behavior and the properties of the domain where the system to be operated is essential in such software development process.

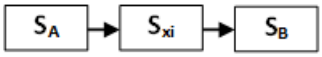
In safety-critical system development, the main focus is on how the system failures and the unexpected system behavior can lead to risk the safety. However, very little research is conduct on how the human subjects can misuse the system or how to handle the unexpected human behavior. Most of the safety-critical systems are unable to handle or even detect users' unexpected behaviors. Hence, the computational model to understand the human behavior has significant applications in safety-critical systems.

V. Appendix A: Raw Observational Data Collection Approaches

- **Sensors** Sensors are devices that can detect or measures physical properties and either record, indicate, or respond to those properties. With the human interest of automation, smart technologies, and the Internet of things these sensors became cheaper and smaller in scale than never before. At present, almost all the devices used by humans in daily basis include some kind of sensors. Moreover, the development of unobtrusive wearable and implantable sensors is also becoming one of the key topics in both academic and industrial areas. Use of sensor records related to human activities and the surrounding environmental factors is a primary way of collecting raw observational data. In addition to being an inexpensive approach, sensor records are more convenient in terms of processing and analyzing.
- **Images, Audio, and Video Clips** Both sound and visual artifacts such as images, audio and video clips related to humans can be used as resources of observation. The successful research studies in image processing, audio processing and video processing areas during the last decade made it possible to extract information from these artifacts effectively and accurately. The extracted information can be used as the observational data that represents both human activities and environmental factors.
- **Social Media** At present most of the humans express their interests, attitudes, and feelings through the social media. According to the literature, use of language is one of the primary ways of humans expressing their mental state. In order to process, analyze and understand the written texts, one can use various natural language processing techniques.
- **Tracking devices and Smart Phones** Nowadays tracking devices such as GPS devices, blood pressure, and heartbeat monitors and smart devices such as smartphones and smart wristwatches are becoming inevitable companions of the humans. The signals transmitting from and to these devices can be used as raw observational data to identify the human activities and conditions of their current locations.

VI. Appendix B: Modification to Situation-Transition Structure Generating Algorithm Using a Sliding Window Approach

The problem of losing information on indirect situation transition in Algorithm 3 can be graphically explained as follows.



Suppose situation S_A is indirectly transit to situation S_B through some other situations S_{xi} in most of the observations, however, neither the transit from S_A to S_{xi} nor from S_{xi} to S_B is significant. In such case, the proposed Situation-transition structure generating algorithm given in Algorithm 3 will be failed to identify the indirect transition from S_A to S_B.

However, this information loss can be minimized by modifying the Algorithm 3 such that it will use sliding window approach to find the mutual information values of the indirect situation transitions between the current situation and the future situation. The distance between the current situation and the far most future situation to be considered depends on the size of the window selected and measured using the number of intermediate situations in between. The modified Algorithm 3 is given in as Algorithm 4.

Note that the proposed solution will increase both the time and space complexity of the Situation-transition structure generating algorithm. Finding the better solution will be left as a future work.

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.

Data: SITU_SEQ : Sequence of situations with time, SITU_CON : Situation-concurrency set, W: Window size, T : Threshold value
Result: Situation transition structure
Read the Input datasets
num_Transitions: Number of situation transitions in SITU_SEQ. Initialize to zero.
SITU_AVAILABILITY = Mapping of each elements in SITU_CON with a $Wx2$ integer array of elements. All elements initialize to zero.
SITU_PAIR_RELATION = Mapping of each possible pair of elements in SITU_CON with a $Wx4$ integer array of elements. All elements initialize to zero.

```
                                                                    /* Defining mappings */
```

for *each x in SITU_CON* **do**
    a = $W \times 2$ Integer array. All elements initialize to zero.
    /* $\forall isuchthat 0 \leq i < W$ */
    /* a[i][0]=Number of occurences where x does not apper in a pair of nodes with i nodes in between in SITU_SEQ */
    /* a[i][1]=Number of occurences where x appers in a pair of nodes with i nodes in between in SITU_SEQ */
    Map x with a in SITU_AVAILABILITY
    for *each y in SITU_CON* **do**
        b = $W \times 4$ Integer array. All elements initialize to zero. /* $\forall jsuchthat 0 \leq j < W$ */
        /* b[j][0]=Number of occurences where x,y is not the current pair of situations with i situations in between in SITU_SEQ */
        /* b[j][1]=Number of occurences where x is not the first appeared situation but y is the second appeared situation with i situations in between in SITU_SEQ */
        /* b[j][2]=Number of occurences where x is the first appeared situation but y is not the second appeared situation with i situations in between in SITU_SEQ */
        /* b[j][3]=Number of occurences where x,y is the current pair of situations with i situations in between in SITU_SEQ */
        Map $< x, y >$ pair with b in SITU_PAIR_RELATION
    end
end
current_parent = First situation in SITU_SEQ
for *each current_child array with W situations in SITU_SEQ starting from second situation* **do**
    for *i from 0 to W-1* **do**
        b = Integer array mapped to $< current\_parent, current\_child[i] >$ pair with i situations in between in SITU_PAIR_RELATION
        b[i][3] = b[i][3] + 1
        for *each p in SITU_CON* **do**
            if *p ≠ current_parent and p ≠ current_child[i]* **then**
            end
            otherPB = Integer array mapped to $< p, current\_child[i] >$ pair with i situations in between in SITU_PAIR_RELATION
            otherPB[i][1] = otherPB[i][1] + 1
        end
        for *each c in SITU_CON* **do**
            if *c ≠ current_parent and p ≠ current_child[i]* **then**
                otherCB = Integer array mapped to $< current\_parent, c >$ pair with i situations in between in SITU_PAIR_RELATION
                otherCB[i][2] = otherCB[2] + 1
            end
        end
    end
    num_Transitions = num_Transitions + 1
    /* Assign current_child[0] as the current_parent for the next iteration */
    current_parent = current_child[0]
end

Let a, b be an integer array.
for *each x in SITU_CON* **do**
  for *each y in SITU_CON* **do**
    if $x \neq y$ **then**
      b = Integer array mapped to $< x, y >$ pair in SITU_PAIR_RELATION
      for *i from 0 to W-1* **do**
        $$b[i][0] = num\_Transitions - \sum_{p=1}^{3} b[i][p]$$
      end
    end
  end
  a = Integer array mapped to x in SITU_AVAILABILITY
  for *i from 0 to W-1* **do**
    a[i][0] = b[i][0] + b[i][1]
    a[i][1] = b[i][2] + b[i][3]
  end
end
/* Generate situation dependency structure */
for *each x in SITU_CON* **do**
  Create a node in the graph.
end
for *each x in SITU_CON* **do**
  ax = Integer array mapped to x in SITU_AVAILABILITY
  for *each y in SITU_CON* **do**
    ay = Integer array mapped to y in SITU_AVAILABILITY
    for *each i from 0 to W-1* **do**
      x0 = ax[i][0]
      x1 = ax[i][1]
      y0 = ay[i][0]
      y1 = ay[i][1]
      if $x \neq y$ **then**
        b = Integer array mapped to $< x, y >$ pair in SITU_PAIR_RELATION
        xy00 = b[i][0]
        xy01 = b[i][1]
        xy10 = b[i][2]
        xy11 = b[i][3]
        /* Calculate mutual information of <x,y> */
        if *x0>0 and x1>0 and y0>0 and y1>0 and xy00>0 and xy01>0 and xy10>0 and xy11>0* **then**
          $$MI = \sum_{p,q=0,1} xypq \times ln(\frac{xypq}{xp \times yq})$$
          if $MI \geq T$ **then**
            Create a directed edge from node x to y and label i
          end
        end
      end
    end
  end
end

Algorithm 4: Modified situation-transition structure generating algorithm using a sliding window approach.

[2] 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.
[3] J. Han, H. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, Proc. Conf. on the Management of Data, 2000, pp. 1 – 12.
[4] U. M. Fayyad and K. B. Irani, “Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning”, Proc. 13th Int. Joint Conf. on Artificial Intelligence, 1993, pp. 1022 – 1027.
[5] J. Pearl, “Probabilistic reasoning in intelligent systems: networks of plausible inference”, Morgan Kaufmann Publishers Inc, 1988.
[6] C. Glymour and G.F. Cooper, “Computation, Causation, and Discovery”, MIT Press, 1999.
[7] 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.
[8] M. Hua, “Situf: a domain specific language and a first step to-wards the realization of situ framework”, Doctoral Dissertation, Iowa State University Ames, IA, USA, ISBN: 978–1–267–63604–1, 2012.
[9] C. Robson, “Real World Research (2nd ed.)”, Wiley-Blackwell, 2002.
[10] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy and J. Albrecht. , “Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes”. Proc. Workshop on Data Mining Applications in Sustainability, 2012.
[11] “CoRE: Situation Centric Intention Driven Research Experiment for Requirements Engineering and Intrusion Detection”, IRB ID 14–347, Iowa State University.
[12] F. Dalpiaz, P. Giorgini and J. Mylopoulos, “Adaptive socio-technical systems: a requirements-based approach”, Requirements Engineering, vol. 18, no. 1, pp. 1–24, 2013.
[13] F. Dalpiaz, E. Paja and P. Giorgini, “Security Requirements Engineering: Designing Secure Socio-Technical Systems”, The MIT Press, 2016.