Mining Hierarchical Scenario-Based Specifications

David Lo
School of Information Systems
Singapore Management University
Email: davidlo@smu.edu.sg

Shahar Maoz
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
Email: shahar.maoz@weizmann.ac.il

***Abstract*—Scalability over long traces, as well as comprehensibility and expressivity of results, are major challenges for dynamic analysis approaches to specification mining. In this work we present a novel use of object hierarchies over traces of inter-object method calls, as an abstraction/refinement mechanism that enables user-guided, top-down or bottom-up mining of layered scenario-based specifications, broken down by hierarchies embedded in the system under investigation. We do this using data mining methods that provide statistically significant sound and complete results modulo user-defined thresholds, in the context of Damm and Harel's live sequence charts (LSC); a visual, modal, scenario-based, inter-object language. Thus, scalability, comprehensibility, and expressivity are all addressed.**

Our technical contribution includes a formal definition of hierarchical inter-object traces, and algorithms for 'zooming-out' and 'zooming-in', used to move between abstraction levels on the mined specifications.

An evaluation of our approach based on several case studies shows promising results.

I. Introduction

Specification mining methods, which automatically infer a system's specification from its execution traces, have attracted much research efforts in recent years [1]–[6]. The mined specifications, whether automata, likely invariant properties, or scenarios, are aimed at aiding in program comprehension and analysis tasks, specifically in the absence of up-to-date documented specifications.

Scalability over long traces, as well as comprehensibility and expressivity of results, are major challenges for specification mining methods. Thus, various filters or abstractions were suggested to address these challenges; e.g., removing utility functions [7], limiting the trace to a predefined fixed set of events of interest [5], nesting depth [8], random sampling [9], considering triggers and effects [10], etc.

In this work we concentrate on *scenario-based specification mining* [6], [10], [11], and introduce the use of a novel abstraction mechanism over event traces, based on hierarchies inherent in the architecture of the system under investigation. Specifically, we consider *inter-object* method calls as events. Given a hierarchy over the objects in the system under investigation, the concept of inter-object becomes relative, and depends on the level of abstraction chosen. An event may be *inter*-object in one level of abstraction and *intra*-object (and thus filtered out) when considered in the context of a higher level. Thus, given a concrete execution trace, a hierarchy of abstract traces is created above it. In the maximal abstract trace, only events exchanged between top-level objects are considered.

We consider scenario-based specification mining at the level of abstraction defined by the user. Further, once a scenario-based specification is mined at a certain level of abstraction, our method allows the user to drill-up and down the hierarchy, practically zooming-out and in on the mined behavioral specifications of the system under investigation.

As a concrete example for a hierarchy that is inherent in the architecture of the system under investigation, we use the Java packages hierarchy. Other hierarchies may be considered, see the discussion in Sec. VI.

The main contribution of our work is in showing how specification mining may take advantage of inherent architectural hierarchies in order to produce a scalable, top-down or bottom-up user-guided abstraction/refinement interactive specification mining method. That is, static program knowledge is used to support stepwise program comprehension through different levels of abstraction. We believe object hierarchies are a useful and natural abstraction and organization mechanism in many architectures, and thus their application to execution traces and specification mining methods is promising. The larger the system under investigation, the greater need arises for an hierarchical organization mechanism of its structure and its specification.

The technical contribution of our work includes the definition and implementation of inter-object event trace hierarchies, the extended mining algorithms, and the algorithms for zooming-out and zooming-in over the hierarchical scenario-based specifications.

In our previous work [6] we used a data mining method for scenario-based specification mining, extracting statistically significant behavioral specifications in the form of a UML2-compliant variant of Damm and Harel's *live sequence charts* (LSC) [12], [13]. Scalability was a major challenge of this work. The key advantages of our current approach over the previous one are (1) incremental, scalable performance, and (2) intuitive, top-down layered results, broken down by hierarchies embedded in the system under investigation. Thus, both scalability and comprehensibility are improved. Recent work on hierarchies in the context of scenario-based programming [14] and the UML2 standard's support for interaction refinement using lifelines *PartDecomposition*, further motivate our work. As in [6], we consider the visual aspect of our work to be an important accessibility factor for engineers; the scenarios mined can potentially be viewed in any UML2-compliant tool.

We have implemented our ideas and evaluated them using a number of case study applications; see Sec. V. The examples throughout the paper are taken from one of these, an instant messaging application called Jeti [15].

The paper is organized as follows. Sec. II presents background material on LSC, scenario-based specification mining, and hierarchical structures in general. Sec. III introduces hierarchical scenario-based specification mining with formal definitions and examples. The algorithms used in our work are described in Sec. IV. Sec. V presents the results of case studies we have conducted in order to evaluate our ideas. Sec. VI discusses some advanced issues of our work, Sec. VII considers related work, and Sec. VIII concludes.

II. Background

We provide background material on live sequence charts, scenario-based specification mining, and system's hierarchies.

A. Live sequence charts

Live sequence charts (LSC) [12], [13] extend classical sequence diagrams with a universal interpretation and must/may modalities. They thus allow to specify scenario-based temporal invariants describing interactions between system objects. The language has been used in the context of execution, verification, and synthesis (see, e.g., [16]–[18]). A translation of LSC into temporal logics appears in [19]. A trace-based semantics for a UML2-compliant variant of LSC appears in [13]. We use a subset of the language, with total ordered messages.

An LSC is composed of two basic charts: a *pre-chart* and a *main-chart*. A basic chart is a tuple $C = (C_L, C_E, C_<)$ where C_L is set of lifelines representing system objects, C_E is a set of inter-object events involving the objects represented by the lifelines in C_L, and $C_<$ is a total order on C_E. Thus, a chart can also be represented as a chain of events $\langle e_1, \ldots, e_n \rangle$. We denote an LSC by $L(pre, full)$, where pre is the pre-chart and $full$ is the concatenation of the pre-chart and main-chart.

Syntactically, lifelines are drawn using vertical lines. Inter-object events are drawn using horizontal arrows from caller to callee; pre-chart events use dashed blue lines, main-chart events use solid red lines.

Semantically, an LSC specifies a temporal invariant: whenever the events in the pre-chart occur in the specified order, eventually the events in the main-chart must occur in the specified order. An LSC does not restrict events not appearing in it to occur or not to occur during a run.

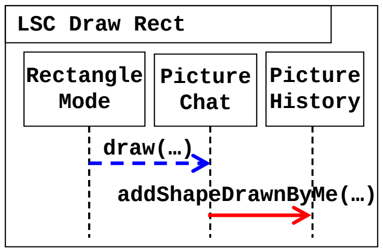

Figure 1. Example LSC: Draw Rectangle

Fig. 1 shows an example LSC. Roughly, this LSC means that *'whenever an object of class* `RectangleMode` *calls the* `draw()` *method of a* `PictureChat` *object, eventually the* `PictureChat` *will call the* `addShapeDrawnByMe()` *method of a* `PictureHistory` *object'*.

B. Scenario-based specification mining

Scenario-based specification mining [6], [10], [11] is concerned with extracting statistically significant LSCs from inter-object traces of a system under investigation.

Inter-object trace, event. A *concrete inter-object trace* is a sequence of inter-object events. A *concrete inter-object event* ev is a tupple $\langle el_1, el_2, m \rangle$ representing an object el_1 (the caller) calling method m of object el_2 (the callee); we require that $el_1 \neq el_2$.

We define the significance of an LSC based on its occurrences in the traces. It is gauged using support and confidence, commonly used metrics in data mining. Below we recall the concepts of scenario instance, positive and negative witnesses, support, and confidence, defined in [6].

Chart instance. Satisfaction of a chart follows the semantics of LSC. We refer to a sub-trace (or a segment of consecutive events in the trace) satisfying the chart C as an instance of C. A segment of a trace is said to be an instance of a chart C if it obeys the ordering specified by C. Each event in the chart must map to a corresponding event in the segment appearing in the specified order. Other events not specified by the chart can occur in any order unrestrictedly.

To describe an LSC chart instance, we use the following Quantified Regular Expressions (QRE) [20]. A quantified regular expression is very similar to standard regular expression with ';' as concatenation operator, '[-]' as exclusion operator (*i.e.* [-P,S] means any event except P and S) and * as the standard kleene-star. The formal definition of an instance of a chart is given in Defn. 2.1 (c.f., [6]):

*Definition 2.1 (***Instance of a Concrete Chart)***:* Given a concrete chart $C = (C_L, C_E, C_<)$, a trace segment $SB = \langle sb_i, sb_{i+1}, \ldots, sb_{i+m-1} \rangle$ is an instance of C if SB follows the QRE expression

$$e_1; [-G]*; e_2; \ldots; [-G]*; e_n \qquad where,$$

$C_E = \{e_1, e_2, \ldots, e_n\}$, $\forall_{0<i<n}. e_i <_C e_{i+1}$, and $G = C_E$.

Fig. 2 shows a short sample from an inter-object trace. The trace includes 3 instances of the LSC shown in Fig. 1: $I_1 = \langle 1, 2, 3\rangle$, $I_2 = \langle 5, 6\rangle$, $I_3 = \langle 9, 10\rangle$.

```
1  RM PC            draw()
2  PC Backend       getMyJID()
3  PC PH            addShapeDrawnByMe()
4  Backend Connect  send()
5  RM PC            draw()
6  PC PH            addShapeDrawnByMe()
7  Backend Connect  send()
8  RM PC            draw()
9  RM PC            draw()
10 PC PH            addShapeDrawnByMe()
```

Figure 2. Part of a sample trace (`RM` stands for `RectangleMode`, `PC` for `PictureChat`, `PH` for `PictureHistory`; the actual trace includes the full qualified name and signatures of the classes and methods involved).

Witnesses. Based on the above definition of a chart instance, we define the notion of positive and negative witnesses of an LSC. Recall that an LSC is composed of a pre-chart and a main-chart. A *positive witness* of an LSC L = $L(pre,full)$, is a trace segment satisfying the $full$ chart – by extension the pre chart as well, since pre is a prefix of $full$. A *negative witness* of L is positive witness of pre which can not be extended to a positive witness of L (or $full$). We say that a negative witness is a *weak negative witness* if the positive witness of pre cannot be extended due to end-of-trace being reached (see discussion in [6]).

Support & Confidence. We use the above notions of witnesses to define the statistical *support* and *confidence* metrics for LSC. Support and confidence are commonly used statistics in data mining [21]. Given a trace T, the *support* of an LSC L= $L(pre, full)$, denoted by $sup(L)$, is simply defined as the number of positive witnesses of $full$ found in T. The *confidence* of an LSC L, denoted by $conf(L)$, measures the likelihood of a sub-trace in T satisfying L's pre-chart, to be extended such that L's main-chart is satisfied or the end of the trace is reached. Hence, confidence is expressed as the ratio between the number of positive-witnesses and weak-negative-witnesses of the LSC and the number of positive-witnesses of the LSC's pre-chart:

$$conf(L, T) \equiv_{def} \frac{|pos(full,T)|+|w\_neg(full,T)|}{|pos(pre,T)|}$$

Notation-wise, when T is understood from the context, it can be omitted.

The support metric is used to limit the extraction to frequently observed interactions. The confidence metric restricts mining to such pre-charts that are followed by particular main-charts with high likelihood. In scenario-based specification mining we are interested in mining statistically significant LSCs: those which occur frequently in the trace (have high support) and in which the pre- is followed by the main- chart with high likelihood (have high confidence). A chart is said to be *significant* if it obeys minimum thresholds of support and confidence, denoted by min_sup and min_conf respectively.

For the LSC shown in Fig. 1 and the trace shown in Fig. 2, $supp(L) = 3$, $conf(L) = 3/4$.

Data mining algorithms to compute a statistically sound and complete set of LSCs, given a trace (or a set of traces) and thresholds for minimal support and confidence, were presented in [6]. These were extended in [11], to handle symbolic scenario-based specifications (at the class level rather than the object level), and in [10], to handle the special case of trigger and effect mining.

C. System's hierarchies

Hierarchical structures are used as an organizing principle in many systems' architectures. Different hierarchies may be used, e.g., the 'part-of' composition hierarchy of components and sub-components, the 'is-a' inheritance hierarchy of classes and subclasses, etc. They are used during systems' development as a way to cope with complexity and size, enabling, e.g., division of labor and reuse.

Specifically, we use here the Java packages hierarchy. Formally, this defines a partial order on classes and packages. An example of such a hierarchy is given in Fig. 3.

```
jeti.
     backend.
             Jabber
     jabber.
             Backend
     plugins.
             titlescroller.
                           Plugin
             titleflash.
                        Plugin.
                               Flash
     ui.
        ChtWndw
        ChtWndws
        ChtSplitPane
```

Figure 3. Part of Jeti's packages hierarchy.

III. Hierarchical Scenario-Based Specification Mining

We are now set out to present hierarchical scenario-based specification mining. We define a hierarchy over inter-object traces and LSCs, and show how hierarchical information is incorporated to enable an interactive mining experience. We use formal definitions and examples.

A. Hierarchy, abstraction, and traces

Hierarchy. Given a system S made of elements $el_1, el_2, \ldots$, a *hierarchy* h is defined by a partial order $\preceq_h$ on its elements. In our specific setting, system elements are Java packages and classes; h is defined using the inclusion relation between packages and classes in the system under investigation.

Abstraction map. An *abstraction map* is a map $amap$: $S \rightarrow S$, which maps elements of S into higher level elements: $amap(el_1) = el_2$ entails $el_1 \preceq el_2$. Note that two elements of S may be mapped by $amap$ to the same higher level element.

The abstraction map is used to define the user's level of abstraction of interest.

Abstract inter-object events. An *abstract inter-object event* ev is a tuple $ev = \langle el_1, el_2, m\rangle$ representing element el'_1 (the caller) calling method m of object el'_2 (the callee) where $el_1 \neq el_2$, $el'_1 \preceq_h el_1$, and $el'_2 \preceq_h el_2$. We refer to el_1 and el_2 as the abstract caller and callee of ev, respectively.

We extend the relation $\preceq_h$ from elements to events as follows: for $ev_1 = \langle el_1, el_2, m_1\rangle$ and $ev_2 = \langle el_3, el_4, m_2\rangle$, $ev_1 \preceq_h ev_2$ iff $el_1 \preceq_h el_3$, $el_2 \preceq_h el_4$, and $m_1 = m_2$.

The abstraction map $amap$ is extended in a natural way from elements to events too. If $amap(el_1) = el'_1$ and $amap(el_2) = el'_2$ then $amap(\langle el_1, el_2, m\rangle) = \langle el'_1, el'_2, m\rangle$.

A hierarchy over inter-object traces. The $\prec_h$ relation and map $amap$ are extended from events to inter-object traces. However, high-level events that are no longer inter-object, i.e., where the abstract caller and the abstract callee are equal, are removed from the high-level trace. That is, for each event $\langle el_1, el_2, m\rangle$ in the trace, s.t. $amap(el_1) = el'_1$ and $amap(el_2) = el'_2$, if $el'_1 \neq el'_2$ then $amap(\langle el_1, el_2, m\rangle) = \langle el'_1, el'_2, m\rangle$; otherwise, if $el'_1 = el'_2$, $amap(\langle el_1, el_2, m\rangle) = \perp$.

Given a hierarchy over the objects in the system under investigation, the concept of inter-object becomes relative, and depends on the level of abstraction chosen, as defined by $amap$. An event may be *inter*-object in one level of abstraction and *intra*-object (and thus filtered out from the inter-object trace) when considered in the context of a higher level. Thus, given a concrete execution trace and an abstraction map, an abstract inter-object trace is created above it.

B. High-level scenarios

Given a hierarchy h, we define high-level LSCs as follows.

High-level scenarios. High-level LSCs may include lifelines representing high-level elements, e.g., packages. Events specified in high-level LSCs are abstract inter-object events.

A refinement relation between LSCs. An LSC L_1 refines an LSC L_2, denoted by $L_1 \preceq_h L_2$, iff

- each lifeline in L_1 has a corresponding high-level lifeline in L_2: $\forall l_1 \in L_1 \exists l_2 \in L_2$ s.t. $l_1 \preceq_h l_2$,
- each event in L_2 has a corresponding lower-level event in L_1: $\forall ev \in L_2 \exists ev' \in L_1$ s.t. $ev' \preceq_h ev$,
- the partial order between corresponding events is preserved: $\forall ev_a, ev_b \in L_2$ s.t. $ev_a <_{L_2} ev_b$, $\exists ev'_a, ev'_b \in L_1$ s.t. $ev'_a \preceq_h ev_a \wedge ev'_b \preceq_h ev_b \wedge ev'_a <_{L_1} ev'_b$.

For example, the LSC shown in Fig. 4 (left) refines the LSC shown in Fig. 4 (right). Note that the order of corresponding events is preserved between the two charts. Also note that the event `send` between `backend.Connect` and `backend.Output` in `Send Packet [refined]` does not have a corresponding event in `Send Packet [abs]`; at the level of abstraction defined by the lifelines of the latter, `send` is intra-object, and is thus not included in this higher-level LSC.

High-level chart instance. Defn 3.1 defines an instance of a high-level chart. It follows as a natural extension of the semantics of an LSC concrete chart given in Defn 2.1.

*Definition 3.1 (***Instance of a High-Level Chart):** Given a high-level chart $C = (C_L, C_E, C_<)$ and an abstraction map $amap$, a trace segment $SB = \langle sb_i, sb_{i+1}, \ldots, sb_{i+m-1}\rangle$ is an instance of C if SB follows the QRE expression

$$e_1; [-G]*; e_2; \ldots; [-G]*; e_n$$

where $C_E = \{amap(e_1), amap(e_2), \ldots, amap(e_n)\}$, $\forall_{0<i<n}.\ amap(e_i) <_C amap(e_{i+1})$, and $G = \{e | amap(e) \in C_E\}$.

Witnesses, support and confidence. Based on the above definition of high-level LSC instance, the measures of positive and negative witnesses, support and confidence could be calculated based on the description in Sec. II. Only those LSCs whose support and confidence are above the respective threshold of min_sup and min_conf are defined as significant.

C. Zoom-out and zoom-in

Given an LSC, two operations are defined to assist the engineer in moving between levels of abstraction and investigating the specification.

Zoom-out. Zoom-out operation abstracts an LSC higher up in the hierarchy. In the process, some lifelines are relabeled and some inter-object events becomes intra-object and are abstracted away. Formally, given an LSC L_1 and an abstraction map $amap$, the operation returns L_2 such that $L_1 \prec_h L_2$ and $\forall l_1 \in L_1 \exists l_2 \in L_2$ s.t. $l_1 \preceq_h l_2$ and $amap(l_1) = l_2$. The operation is denoted by $zout(L_1)$.

Zoom-in. Zoom-in operation adds details to a higher level LSC. It moves a higher level LSC down in the hierarchy. Some intra-object events become inter-object and are added to the LSC. Formally, given an LSC L and an abstraction map $amap$, the operation returns the set of all LSCs L_i such that $L_i \preceq_h L$ and $\forall l_{ij} \in L_i \exists l \in L$ s.t. $l_{ij} \preceq_h l$ and $amap(l_{ij}) = l$. The operation is denoted by $zin(L_1)$.

Some examples of zoom-out and zoom-in operations are given in Fig. 4 and Fig. 5 respectively.

Most importantly, in the context of specification mining, the refinement relation $\preceq_h$ on LSCs and the zoom-in/out operations defined above must be revised to include the statistical support and confidence of the LSCs on the given trace. Intuitively, if $L_1 \preceq_h L_2$, $sup(L_1)$ may be lower than $sup(L_2)$: since L_2 is more abstract it may have instances that are not instances of L_1. Thus, the revised definitions for the zoom operations are as follows:

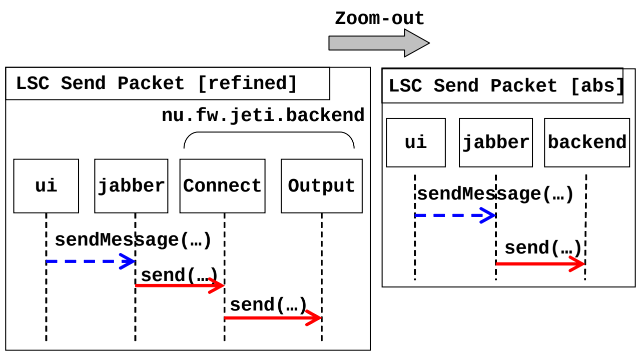

Figure 4. Zoom-Out

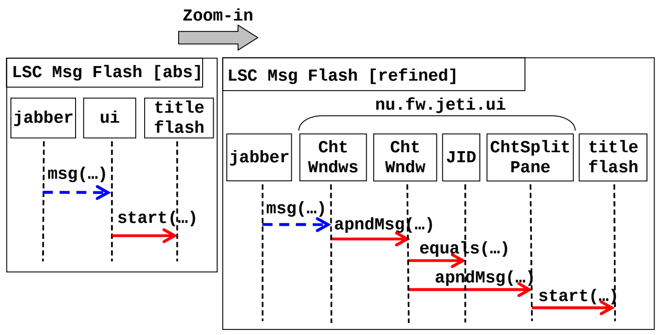

Figure 5. Zoom-In

Statistical zoom-out. Given an LSC L_1 and an abstraction map $amap$, the operation returns L_2 such that $zout(L_1) = L_2$, $sup(L_1) = sup(L_2)$, and $conf(L_1) = conf(L_2)$. We denote this operation by $szout(L_1)$.

Statistical zoom-in. Given an LSC L_1 and an abstraction map $amap$, the operation returns a set of LSCs $\{L_2 | L_2 \in zin(L_1) \wedge sup(L_1) = sup(L_2) \wedge conf(L_1) = conf(L_2)\}$. We denote this operation by $szin(L_1)$.

Unless otherwise stated, for the remainder of the paper, we use zoom-in and zoom-out to refer to statistical zoom-in and zoom-out respectively. The algorithms to compute the above are given in the next section.

IV. Framework & Algorithms

In this section we present an overview of our framework. We then describe the algorithm to mine a set of statistically significant LSCs at the selected abstraction level and the algorithms to compute the zoom-out and zoom-in operations.

A. *Mining framework*

Hierarchical scenario-based specification mining starts with a hierarchy and a concrete inter-object trace. Given these inputs and user-defined thresholds of support and confidence, LSCs at various abstraction levels are mined interactively, following the steps described below.

```
Procedure ZoomOut
Inputs:
    $T$ : Input trace
    $amap$: Abstraction map
    $L(pre, full)$: Input LSC
    $pre = \langle e_1, \ldots, e_n \rangle$: L's pre-chart
    $full = \langle e_1, \ldots, e_m \rangle$: L's pre- & main- chart
Output:
    A corresponding high level LSC
Method:
1: Let $ts = sup(L(pre, full))$
2: Let $tc = conf(L(pre, full))$
3: Let $pre' = \langle e'_1, \ldots, e'_n \rangle$, where $\forall_i . e'_i = amap(e_i)$
4: Let $full' = \langle e'_1, \ldots, e'_m \rangle$, where $\forall_i . e'_i = amap(e_i)$
5: Compute $pos(pre')$, $w\_neg(full')$, and $pos(full')$ on $T$
6: Let $conf = \frac{|pos(full',T)| + |w\_neg(full',T)|}{|pos(pre',T)|}$
7: If $(pos(full',T) = ts \wedge conf = tc)$
8:     Output $L(pre', full')$
```

Figure 6. Zooming-Out

```
Procedure ZoomIn
Inputs:
    $T$ : Input trace
    $amap$: Abstraction map
    $L(pre, full)$: Input LSC
    $pre = \langle e_1, \ldots, e_n \rangle$: L's pre-chart
    $full = \langle e_1, \ldots, e_m \rangle$: L's pre- & main- chart
Output:
    A set of corresponding lower level LSCs
Method:
1:  Let $ts = sup(L(pre,full))$, Let $tc = conf(L(pre,full))$
2:  Let $preS = \{\langle e'_1, \ldots, e'_n \rangle \mid \forall_i.\ amap(e'_i) = e_i\}$
3:  Let $fullS = \{\langle e'_1, \ldots, e'_m \rangle \mid \forall_i.\ amap(e'_i) = e_i\}$
4:  Let $SIG = \emptyset$
5:  For each pair $p \in preS$ and $f \in fullS$, where $p$ is
    a prefix of $f$
6:      Let $conf = \frac{pos(f) + w\_neg(f)}{pos(p)}$
7:      If $(pos(f) = ts \wedge conf = tc)$
8:          $SIG = SIG \cup$ AddDetails $(T, L(p,f), amap, ts, tc)$
9: Output LSCs in $SIG$
```

Figure 7. Zooming-In

1) **(Abstraction)** Abstract the concrete trace to a user selected abstraction level defined using an abstraction map. An abstract inter-object trace is created.
2) **(Mining)** Mine high-level LSCs from the abstract inter-object trace. A set of statistically significant high-level LSCs is mined.
3) **(Selection)** The user chooses a subset of the mined LSCs for further investigation.
4) **(Zoom-out, zoom-in, and event filtering)**
    a) Zoom-out. Based on a new abstraction map, the user instructs the miner to abstract a selected LSC further up the hierarchy. Some inter-object

```
Procedure AddDetails
Inputs:
    $T$ : Input trace
    $L(pre',full')$: Lower level LSC to be expanded
    $amap$: Abstraction map
    $ts$: Support of L, $tc$: Confidence of L
Output:
    Maximal lower-level LSCs corres. to $L(pre',full')$
Method:
1:  Let $EVS = \{e|e \in T \wedge pos(e) \geq ts\}$
2:  Let $SIG = \{0,L(pre',full')\}$
3:  For (int i=0;i<$|full'|$;i++)
4:      For every $(off,L(pre,full))$ in $SIG$
5:          InsertAtLoc $(T,pre,full,\text{i},off,EVS,SIG,ts,tc)$
6:  Remove non maximal LSCs in $SIG$
7:  Output LSCs in $SIG$

Procedure InsertAtLoc
Inputs:
    $T$ : Input trace
    $pre$, $full$: Pre-chart, Full-chart
    $i_O$ : Location to insert in original chart
    $off$: Offset (due to events being added)
    $EVS$: Set of frequent single-events
    $SIG$: Temporary set of lower-level LSCs with offsets
    $ts$, $tc$: Support and confidence of input LSC
Output:
    $SIG$ contains an updated set of lower-level LSCs
Method:
8:  For each $ev$ in $EVS$ not in $full$
9:      Let $loc = i_O+off$
10:     Let pre' = Insert $ev$ in $pre$ at $loc$, if $|pre'| \leq loc$
11:     Let $full'$ = Insert $ev$ in $full$ at $loc$
12:     Let $conf = \frac{(pos(full',T)+w-neg(full',T))}{pos(pre',T)}$
13:     if $(pos(full',T) = ts \wedge conf = tc)$
14:         Add $(off+1,L(pre',full'))$ to $SIG$
15:         InsertAtLoc$(T,pre',full',i_O,off+1,EVS,SIG,ts,tc)$
```

Figure 8. Add Details Procedure

events become internal (intra-object) and are removed.

b) Zoom-in. Alternatively, based on a new abstraction map, the user instructs the miner to refine a selected LSC by adding lower-level details; some internal (intra-object) events become inter-object and the miner expands the LSCs accordingly.

c) Event filtering. During the interactive mining process, the user may find some events uninteresting. The user could then instruct the miner to remove uninteresting events from a selected LSC and recompute its support and confidence.

Zooming-out/in is done while considering the support and confidence of the resulting LSCs.

The above enables one to mine for specifications at a selected abstraction level of interest. Then, the zoom-in and out operations provide an interactive mechanism allowing to elicit user feedback on the mined specifications. If more details are required, the user may choose to zoom-in. Otherwise, if the specification gets too detailed, the user may choose to zoom-out.

B. Mining LSCs

The basic algorithm for LSC mining given support and confidence thresholds was presented in detail in [6]. Roughly, this involves a search space traversal process to identify significant LSCs. We consider the space of all possible LSCs and navigate this space in a systematic way. The following monotonicity property is used to prune the search space containing insignificant LSCs en masse hence enabling the algorithm to run in a reasonable amount of time.

*Property 1 (***Prefix monotonicity***):* If one chart is a prefix of another chart, the number of positive witnesses of the earlier should be larger than or equal to the number of positive witnesses of the latter. Formally, given charts C = $\langle e_1, e_2, \ldots, e_n\rangle$ and C' = $\langle e'_1, e'_2, \ldots, e'_m\rangle$, if $\forall_{i\in\{1,\ldots,n\}}.\ e_i{=}e'_i$, then $pos(C) \geq pos(C')$.

We first consider charts of length 1, and grow this chart by appending events one by one. Once the number of positive witnesses of the chart is lower than the minimum support threshold, based on Prop. 1, we know that any extension of the chart would not have a higher number of positive witnesses. When this is the case we stop extending the chart and backtrack. When this process ends, we obtain a set of charts whose number of positive witnesses is larger than the minimum support threshold. These basic charts are composed to form LSCs – consisting of a pre-chart and a main-chart. Only LSCs that are significant are outputted (see [6] for details).

In this paper we consider only LSCs with no repeated events. Under this condition, Prop. 1 could be strengthened to Prop. 2 described below. This is used by the algorithm for zoom-in described in subsection IV-D.

*Property 2 (***Sub-chart monotonicity***):* If a chart is a sub-chart of another chart and there are no repeated events in the chart, then the number of positive witnesses of the former is larger or equal than the number of positive witnesses of the latter. Formally, given charts C = $\langle e_1, e_2, \ldots, e_n\rangle$ and C' = $\langle e'_1, e'_2, \ldots, e'_m\rangle$, if $\not\exists_{e_i,e_j\in C'}.\ i{\neq}j \wedge e_i{=}e_j$ and there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ s.t. $\forall_{j\in\{1,\ldots,n\}} e_j = e'_{i_j}$, then $pos(C) \geq pos(C')$.

For the initial LSC mining, given a trace and an abstraction map $amap$, we first compute the abstract inter-object trace induced by $amap$ and then use it as input for the basic LSC mining algorithm of [6], using the semantics of high-level LSC instance. We now focus on the zoom-out and zoom-in operations, which are unique to hierarchical specification mining and are thus part of the technical contribution of the present work.

C. Zooming-out

The zoom-out operation defined earlier takes a mined LSC L_1 and a user-defined abstraction map $amap$, and outputs a higher-level LSC L_2 s.t. $L_1 \preceq_h L_2$ and $\forall l_1 \in L_1 \exists l_2 \in L_2$ s.t. $l_1 \preceq_h l_2$ and $amap(l_1) = l_2$. In the context of mining, we need to find L_2 and compute its support and confidence metrics. L_2 is outputted if it has the same support and confidence as L_1.

Fortunately, computing L_2 and its statistical significance does *not* require re-mining for all significant LSCs from the higher-level trace induced by $amap$. Instead, we first find L_2 from L_1, and then find its positive and negative instances on the trace, to compute its support and confidence.

Finding L_2 from L_1 is trivial: lifelines are mapped according to $amap$, inter-object events that have become intra-object are removed from L_2.

Computing a given LSC's support and confidence on a trace is significantly much cheaper operation than mining all significant LSCs from a trace; its complexity is linear to the length of the trace. No backtracking and thus no exponent is involved in the algorithm.

Pseudo-code summarizing the above appears in Fig. 6.

D. Zooming-in

Zooming-in is more challenging, as it has to return all lower level LSCs that refine the given higher-level LSC and whose support and confidence are equal to its support and confidence. Again the user specifies a mapping $amap$. Based on this mapping, we first refine all high-level lifelines to the corresponding lower-level lifelines. This results in a set of corresponding lower-level LSCs. Zoom-in potentially returns a set rather than a single LSC. Each of the LSCs in the set adds different set of low-level events to the original higher-level events.

Next we add the intra-object events that are inter-object vis-á-vis the more refined lifelines. This poses a computational challenge. Given a lower level LSC $L(pre,full)$ prescribed by $amap$, adding newly introduced inter-object events must be done to the pre and full charts.

We refer to each of these steps as *growing* a chart. Let us illustrate this process with the full chart (the same applies for the pre-chart). We try to insert new events at the various locations or positions in the chart resulting in a new chart $full'$; we then compute $pos(full')$. If $pos(full')< pos(full)$ we stop growing the chart further – based on Prop. 2 – and try to insert at another location. The detailed pre- and full-charts are later composed to form significant LSCs with the same support and confidence as the higher level LSC. We output LSCs containing the maximal number of inter-object events that could be added to the LSC without affecting its support and confidence values.

Pseudo-code summarizing the zoom-in algorithm appears in Fig. 7. This corresponds to the lifelines refinement process. It makes use of the `AddDetails` procedure described in Fig. 8, which weaves in newly introduced inter-object events. The `AddDetails` procedure invokes the `InsertAtPos` procedure, iteratively, trying to insert events at various locations in the LSC.

E. Correctness and Complexity

Below we describe our statistical soundness and completeness guarantee and informally discuss the complexity and performance benefits of using hierarchies in LSC mining.

Statistical soundness and completeness. An algorithm mining significant specifications is *statistically sound and complete* if all mined specifications are significant (sound), and all significant specifications are mined (complete). This notion is commonly used in data mining, and is guaranteed by, e.g., Daikon [22] or data mining applications [21]. Our algorithms are sound and complete modulo the given traces, user-defined thresholds and abstraction level considered (our notion of soundness and completeness is thus independent of the quality of the traces used in terms of coverage etc., that is, in contrast to, e.g., [23]).

Our zoom-in algorithm guarantees that all maximal LSCs refined from the input LSC having the same support and confidence would be outputted. Also, the support of the resulting LSCs will be equal to the support of the input LSC. Our zoom-out algorithm guarantees that the output LSC (if any) has the same support and confidence as the input LSC. If no LSC is found, we know that there is no corresponding higher-level LSC with the same support and confidence.

Complexity. The complexity of the mining algorithm is linear in the number of frequent charts considered (c.f. [6]). However, the higher the abstraction in the hierarchy, the less frequent charts are expected to appear in the trace. This has far reaching consequences on mining times; see Sec. V.

In zoom-in, since only one LSC is zoomed-in to the lower-level search space, the frequent charts that need to be generated are only a small fraction of the total number of frequent charts at the lower abstraction level. These charts are also easy to find as the high-level LSC provides a constraint that restricts the search space to be significantly smaller. The complexity of zoom-out is very small as one needs only scan the trace once.

V. Experiments & Evaluation

We have implemented our ideas and applied them to several applications. We report on two of these below.

Experiments setup. We generated inter-object traces from two open source applications: Jeti [15], a full featured instant messaging application, consisting of 49K LOC, 3400 methods, and 511 classes in 62 packages; and Columba [24], a rich email client, consisting of 46K LOC, 6200 methods, and 1139 classes in 226 packages. Trace generation was

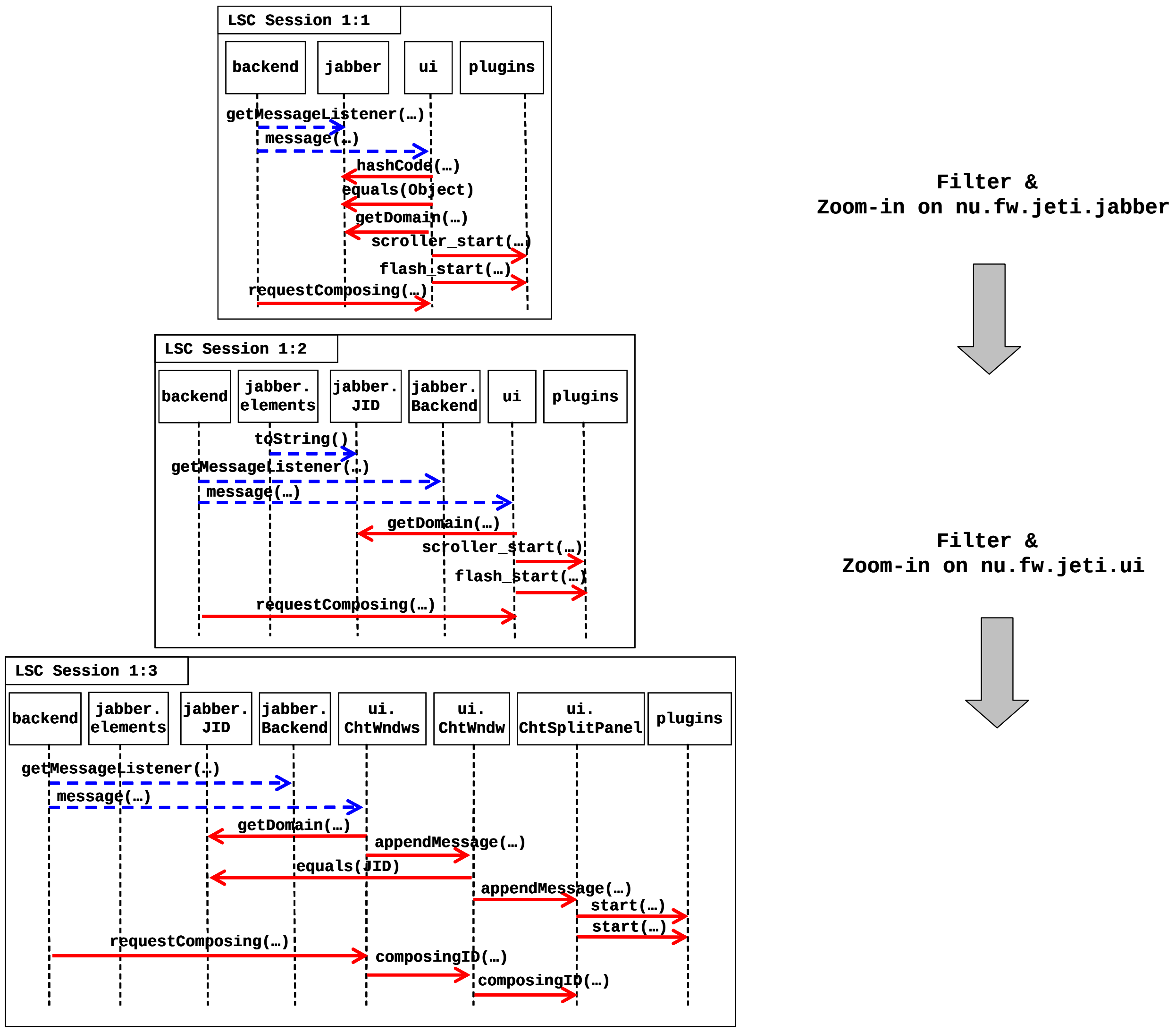

Figure 9. Interactive Mining Session 1: Jeti's Message Receive

done using AspectJ, recording inter-object method calls, in a format similar to the one shown in Fig. 2. We collected traces of length 11,230 and 5,921 events from Jeti and Columba respectively.

Below we describe three interactive hierarchical specification mining sessions, involving mining, zoom-in, zoom-out, and event filtering. We compare our algorithm with previous work on LSC mining [6], which did not utilize hierarchies. All experiments were executed on an Intel Core2 Duo 2.40GHz 3.24GB RAM Windows XP Tablet PC. The algorithms are programmed using C#.Net compiled using VS.Net 2005. Additional results from our experiments, including traces, are available in [25].

Interactive mining session 1. We describe a mining session of Jeti, consisting of several steps. First, we set up an *amap* and mined the LSC shown in Fig. 9 (top) with support 10 and confidence 1. The *amap* used maps every class in the trace to its corresponding package at depth 4, e.g., `nu.fw.jeti.ui.ChatWindow` $\mapsto$ `nu.fw.jeti.ui`. This LSC describes the behavior that takes place when Jeti receives a new message. We filtered out the methods `hashCode()` and `equals()`, as we consider them uninteresting, and then zoom-in on the `jabber` package, resulting in the LSC shown in the middle of Fig. 9. Another filtering on method `toString()` and zoom-in to expand the `ui` package, resulted in the LSC shown in Fig. 9 (bottom). The scenario shows how, after a new message was received, the application gets information about the communicating parties, flashes the title bar, scrolls the incoming message on the title bar, and prepares for composing a reply. The more detailed LSC describes the internal behavior within the `ui` package where the message is appended to the `ChatSplitPanel` and message identifier is set in the user interface's window and panel.

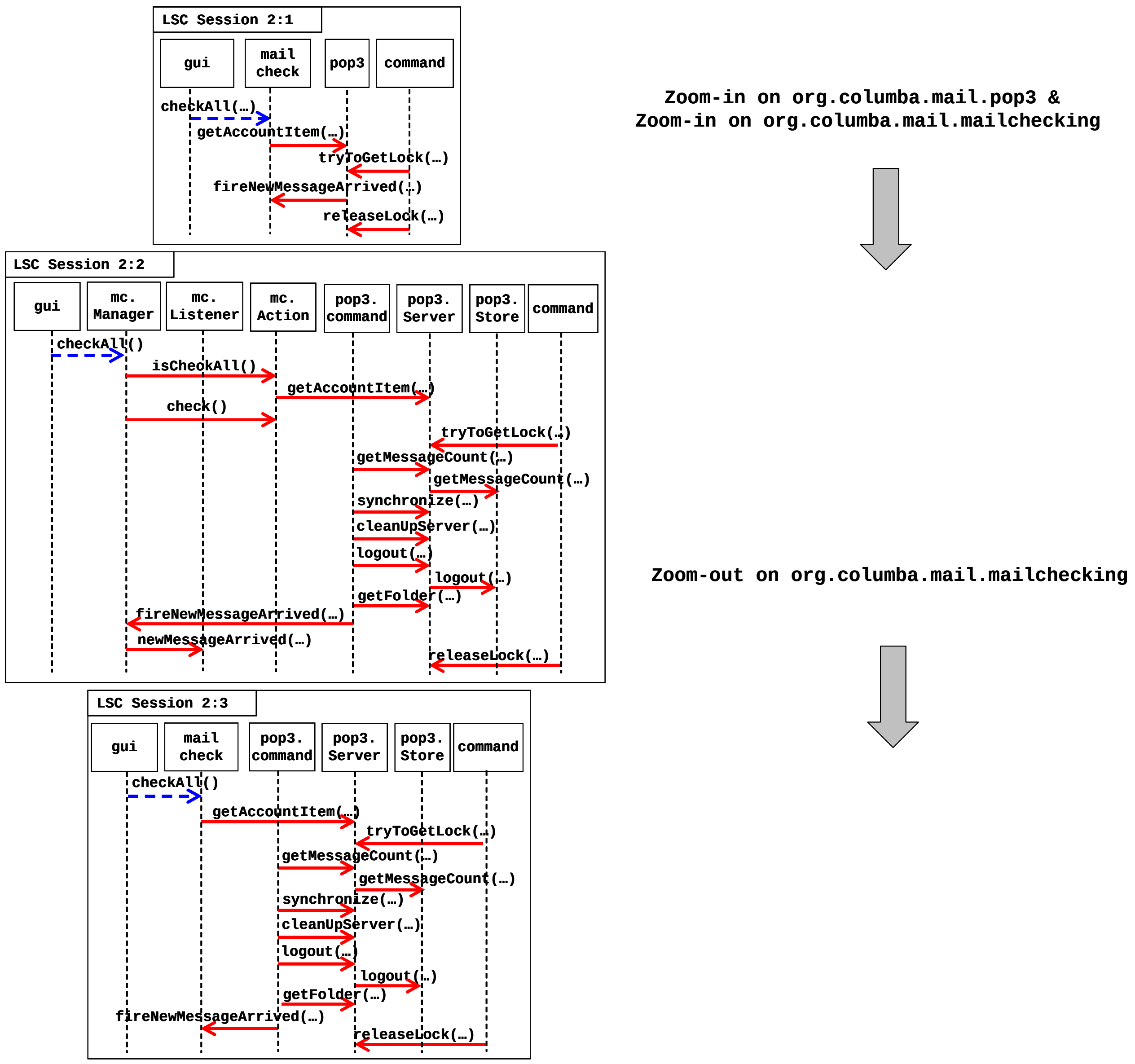

Figure 10. Interactive Mining Session 2: Columba's Check Mail

Interactive mining session 2. Next, we describe a mining session of Columba, consisting of several steps too. First, we set up an *amap* and mined the LSC shown in Fig. 10 (top) with support 10 and confidence 1. Again, the *amap* maps every class in the trace to its corresponding package at depth 4. This LSC describes a scenario of checking for new emails. The system gets account information, locks the server, processes new emails (if any), and finally releases the lock. To get more details about this scenario, we zoomed-in on the `pop3` and `mailcheck` packages. After reviewing the new LSC, we decided that the internal communication within the `mailcheck` package is not of interest. Thus, we zoomed-out from `mailcheck`, arriving at the LSC shown in Fig. 10 (bottom). This LSC gives detailed information regarding the processing of new emails namely: all new messages are retrieved, synchronization is performed, old messages are deleted, logout is performed, and the inbox folder is accessed.

App.	min_sup	No Hier.	Hier.	TRed
Jeti	10	$>$ 10 hrs	2.45 sec	78.94%
	5	$>$ 10 hrs	2.58 sec	
Columba	10	123.31 sec	0.41 sec	11.78%
	5	Out-of-mem	2.58 sec	

Table I
INITIAL MINING SPEED

Interactive mining session 3. Here we started off with the LSC `Send Packet [abs]` shown in Fig. 11 (left). This was not a previously mined LSC, but rather one we suggested as a hypothesis, describing the process of sending a message. First, the miner found that the given LSC support and confidence are 11 and 1, respectively. Second, we

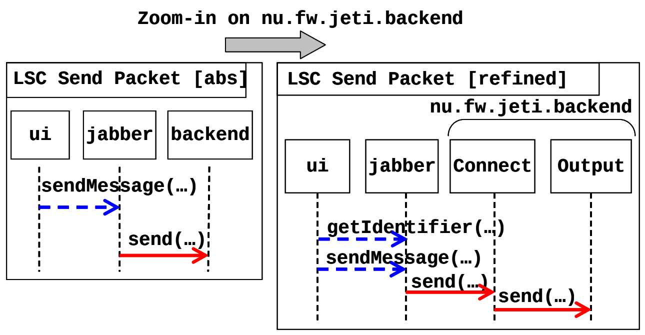

Figure 11. Interactive Mining Session 3: Jeti's Send Message

Session	Filter	Zoom-in	Zoom-out
1	0.02 sec	0.16 sec + 0.23 sec	N/A
2	N/A	0.64 sec + 0.17 sec	0.03 sec
3	N/A	0.17 sec	N/A

Table II
FILTER/ZOOM-IN/ZOOM-OUT SPEED

applied zoom-in to expand on the `nu.fw.jeti.backend` lifeline. The result is shown in Fig. 11 (right); it revealed the internal communication with the `backend` package and another method between the `ui` and `jabber` packages.

The three mining sessions described above show the utility of our work in enabling an interactive user-guided spec. mining process, resulting in valuable knowledge about the systems under investigation, taking advantage of the hierarchical setting to produce expressive and comprehensible results at various levels of abstraction. Next we report on running times.

Performance results. The use of abstraction maps for initial mining resulted in significant reduction in running time. Table I shows a summary of running times. The columns (from left to right) correspond to the program under investigation, the minimum support threshold used, the runtime when no hierarchy and abstraction is utilized (based on the algorithm in [6]), the runtime when a hierarchy is utilized, and the percentage of reduction of trace length due to the abstraction map used. The results show that a speed up of more than 290 times could be achieved in all the cases considered. When running the algorithm in [6] on the Columba dataset at support level 5, the algorithm crashes after running for 1.5 hours due to an out of memory exception.

Moreover, zooming-in and zooming-out work very fast, as summarized in Table II. This is expected, as explained in subsection IV-E.

Lessons learned. While the above experiments show promising results in terms of the quality of scenarios mined and of the scalability of our approach, we have also learned some important lessons about the limitations and challenges faced by our current work, some of which are briefly listed below.

First, using the packages hierarchy works well, but is somewhat limiting. Sometime, we look for some flexibility in defining the hierarchy; say, grouping objects by some custom user-defined criteria, e.g., all classes implementing a selected interface of interest.

Second, automating the definition of $amap$. We envision heuristics that scan the trace and suggest 'best abstraction maps'. For example, given an abstract LSC, find lowest (i.e., most detailed) possible abstraction level where support is still not below threshold.

Finally, the visual aspect and interactive nature of our work is a significant advantage. At the moment, we only have a prototype graphical user interface accepting inputs in the form of mouse clicks and text and outputting a textual representation of mined LSCs. Thus, the full potential of the graphical representation is not fulfilled. We plan to address this by completing an integrated solution where mined LSCs are automatically graphically visualized and allow direct manipulation, e.g., zooming-in by clicking on the lifeline to be expanded.

VI. DISCUSSION

Some issues related to our work warrant a discussion.

Other hierarchies. While we demonstrate the use of the Java packages hierarchy for hierarchical spec. mining, our method is general and may be applied to other hierarchies (e.g., a 'part-of' relation between sub-components and components, an 'is-a' relation between subclasses and classes, etc.). In addition, it may be applied to a user-defined custom hierarchy. This is of interest in case a user has some prior knowledge about the system or is interested in a specific task; e.g., grouping together all implementations of a selected interface, so that mined LSCs show only a single lifeline representing this interface and include only those method calls common to all its implementations, regardless of their location in the packages hierarchy. Finally, one may consider the development of 'ad-hoc hierarchies', derived from the specific features of an input trace. We leave this direction for future work.

Limitations of expressivity. Our choice of LSC as the target formalism of our mining approach is motivated by its expressive power, allowing the specification of universal temporal invariants. It is important to note, however, that due to the mix between liveness and safety in the semantics of LSC, the refinement relation $\preceq_h$ between LSCs indicates neither logical implication nor trace containment. Still, following [14] and the UML2 standard's support for *PartDecomposition*, we believe the relation $\preceq_h$ is indeed valuable, in particular when it is based on a hierarchy that is inherent in the structure of the system under investigation.

Scalability. The scalability of hierarchical spec. mining depends on the level of abstraction chosen. The higher the level of abstraction, the shorter the traces and patterns, and hence mining runs faster (see Sec. V). Thus, we recommend starting with the highest level of abstraction of interest, then zooming-in to lower levels only on selected LSCs, as is deemed necessary for better comprehension.

Object IDs. In the present work, object IDs are abstracted away from the input traces. As discussed in previous work [11], this cannot be done in the general case; thus, we implicitly assume no overlapping LSCs. Relaxing this restriction requires further work, see [11].

Relaxing support requirement during zoom-out and zoom-in. In our current study, we require that output chart(s) of zoom-in and zoom-out have the same support and confidence as the input chart. This requirement could be relaxed to only require that the output charts have support values greater than a certain minimum support threshold. However, this comes at a cost, as the number of frequent charts to be investigated would then increase and so will the running time. In this paper, we focus on the more strict version. The algorithm could be extended to support the more relaxed version. The material in [25] gives the description of the relaxed version and a case study showing that it could be useful and run in a reasonable amount of time.

VII. Related Work

Much work has been published on specification mining. For lack of space in this proceedings we briefly discuss only the ones most relevant to our work.

Reverse engineering of sequence diagrams. Many work suggest and implement different variants of reverse engineering of objects' interactions from program traces and their visualization using sequence diagrams (see, e.g., [26], [27]), which may seem similar to our work. Unlike our work, however, all consider and handle only concrete, continuous, non-interleaving, and complete object-level interactions and are not using aggregations and statistical methods to look for higher level recurring scenarios; the reverse engineered sequences are used as a means to describe single, concrete, and relatively short (sub) traces in full (and thus may be viewed not only as concrete but also as 'existential'). In contrast, we look for universal (modal) sequence diagrams, which aim to abstract away from the concrete trace and reveal significant recurring potentially universal abstract scenario-based specification, ultimately suggesting scenario-based system requirements, broken down by architectural hierarchies embedded in the system under investigation.

Scenario-based specification mining. Scenario-based specification mining was first introduced in [6], which presented a data mining algorithm to compute a statistically sound and complete set of LSCs, given a trace (or a set of traces) and thresholds for minimal support and confidence. These were extended in [11], to handle symbolic scenario-based specifications (at the class level rather than the object level), and in [10], to handle the special case of trigger and effect mining. The present work extends the original work with hierarchical structures, and shows significant improvements in terms of scalability, expressivity, and comprehensibility.

Automata-based specification mining. Most specification miners produce an automaton (e.g., [1]–[3], [28]), and have been used for various purposes from program comprehension to verification. Unlike these, we mine a set of LSCs from traces of program executions. We believe sequence diagrams in general and LSCs in particular, are suitable for the specification of inter-object behavior, as they make the different role of each participating object and the communications between the different objects explicit. Thus, our work is not aimed at discovering the complete behavior or APIs of certain components, but, rather, to capture the way components cooperate to implement certain system features. Indeed, inter-object scenarios are popular means to specify requirements (see, e.g., [29]–[31]). Specifically, the hierarchical extension aids us in coping with the length of the traces, and aids the user of the miner in coping with the complexity of the system under investigation.

VIII. Conclusion & Future Work

We presented hierarchical scenario-based specification mining, which uses object hierarchies over traces of inter-object method calls, as an abstraction/refinement mechanism that enables user-guided, top-down or bottom-up mining of layered scenario-based specifications, broken down by hierarchies embedded in the system under investigation. Our technical contribution includes a formal definition of hierarchical inter-object traces, and algorithms for 'zooming-out' and 'zooming-in', used to move between abstraction levels on the mined specifications.

An evaluation of our approach, based on several case studies, demonstrated its utility and its scalability to long traces and complex, large systems. We believe object hierarchies are a useful and natural abstraction and organization mechanism in many architectures, and thus their application to execution traces and specification mining methods is promising. Our experiments confirm this expectation.

The larger the system under investigation, the greater need arises for an hierarchical organization mechanism of its structure and its specification. Indeed, hierarchical structures are used as an organizing principle in many systems' architectures, during systems' design, development, and deployment, as a way to cope with complexity and size, enabling, e.g., division of labor and reuse. Thus, we expect that further experiments with larger applications, will reveal similar, perhaps even stronger, results.

The work is part of the larger framework of *user-guided specification mining*, where specification mining is viewed as a task oriented iterative interactive process allowing the user to focus on issues of interest and use accumulated knowledge about the application under investigation. It aims to support property discovering tasks for debugging, evolution, runtime monitoring, and formal verification.

Future work on scenario-based specification mining in general includes the extension of our work to cover a larger subset of the LSC language, adding support for partial-order and conditions. Related work in-progress concerns various usages of the mined scenarios in practice, e.g., as suggested in [6], the compilation of the mined LSCs into monitoring *scenario aspects* [18], [32], thus using them for runtime verification of the system under investigation.

More specific to hierarchical scenario-based specification mining, based on the lessons learned listed above, planned future work includes further experiments with different hierarchies, the development of heuristics for semi-automatic computation of 'best abstraction maps' under various criteria, and the design of an integrated graphic user interface that takes advantage of the visual nature of our work.

References

[1] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API Patterns as Partial Orders from Source Code: From Usage Scenarios to Specifications," in *SIGSOFT FSE*, 2007.

[2] G. Ammons, R. Bodik, and J. R. Larus., "Mining Specification," in *POPL*, 2002.

[3] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," in *ICSE*, 2008.

[4] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M.Das, "Perracotta: Mining temporal API rules from imperfect traces." in *ICSE*, 2006.

[5] F. C. de Sousa, N. C. Mendonça, S. Uchitel, and J. Kramer, "Detecting Implied Scenarios from Execution Traces," in *WCRE*, 2007, pp. 50–59.

[6] D. Lo, S. Maoz, and S.-C. Khoo, "Mining Modal Scenario-Based Specifications from Execution Traces of Reactive Systems," in *ASE*, 2007.

[7] A. Hamou-Lhadj and T. Lethbridge, "Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system," in *ICPC*, 2006.

[8] A. Kuhn and O. Greevy, "Exploiting analogy between traces and signal processing," in *ICSM*, 2006.

[9] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, "Bug isolation via remote program sampling," in *PLDI*, 2003, pp. 141–154.

[10] D. Lo and S. Maoz, "Mining Scenario-Based Triggers and Effects," in *ASE*. IEEE, 2008, pp. 109–118.

[11] ——, "Specification mining of symbolic scenario-based models," in *PASTE*. ACM, 2008, pp. 29–35.

[12] W. Damm and D. Harel, "LSCs: Breathing Life into Message Sequence Charts," *J. on Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, 2001.

[13] D. Harel and S. Maoz, "Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams," *Software and Systems Modeling*, vol. 7, no. 2, pp. 237–252, 2008.

[14] Y. Atir, D. Harel, A. Kleinbort, and S. Maoz, "Object Composition in Scenario-Based Programming," in *FASE*, 2008.

[15] "Jeti. Version 0.7.6 (Oct. 2006)." http://jeti.sourceforge.net/.

[16] J. Klose, T. Toben, B. Westphal, and H. Wittke, "Check it out: On the efficient formal verification of Live Sequence Charts," in *CAV*, 2006.

[17] H. Kugler and I. Segall, "Compositional Synthesis of Reactive Systems from Live Sequence Chart Specifications," in *TACAS*, 2009.

[18] S. Maoz and D. Harel, "From multi-modal scenarios to code: compiling LSCs into AspectJ," in *SIGSOFT FSE*, 2006.

[19] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps, "Temporal Logic for Scenario-Based Specifications," in *TACAS*, 2005.

[20] K. Olender and L. Osterweil, "Cecil: A sequencing constraint language for automatic static analysis generation." *IEEE TSE*, vol. 16, pp. 268–280, 1990.

[21] J. Han and M. Kamber, *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2006.

[22] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *TSE*, vol. 27, no. 2, pp. 99–123, 2001.

[23] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *ICSE*, 1994.

[24] "Columba, Java Email Client." http://sourceforge.net/projects/columba.

[25] "Hierarchical LSC Mining - Supplementary Website," http://www.mysmu.edu/faculty/davidlo/hie/hie.html.

[26] "Eclipse Test and Performance Tools Platform," http://www.eclipse.org/tptp/.

[27] D. F. Jerding, J. T. Stasko, and T. Ball, "Visualizing Interactions in Program Executions," in *ICSE*, 1997.

[28] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller, "Mining Object Behavior with ADABU," in *WODA*, 2006.

[29] I. Krüger, "Capturing Overlapping, Triggered, and Preemptive Collaborations Using MSCs." in *FASE*, 2003.

[30] G. Sibay, S. Uchitel, and V. A. Braberman, "Existential live sequence charts revisited," in *ICSE*, 2008, pp. 41–50.

[31] J. Whittle, R. Kwan, and J. Saboo, "From scenarios to code: An air traffic control case study," *Software and Systems Modeling*, vol. 4, no. 1, pp. 71–93, 2005.

[32] D. Harel, A. Kleinbort, and S. Maoz, "S2A: A compiler for multi-modal UML sequence diagrams," in *FASE*, 2007.