Mana n Incons stenc es n an Evo v n pec f cat on

Steve Easterbrook

School of Cognitive and Computing Sciences,
University of Sussex, Brighton, BN1 9QH, UK.
easterbrook@cogs.susx.ac.uk

Bashar Nuseibeh

Department of Computing, Imperial College
180 Queen's Gate, London, SW7 2BZ, UK.
ban@doc.ic.ac.uk

Abstract

In an evolving specification, considerable effort is spent handling recurrent inconsistencies. Detecting and resolving inconsistencies is only part of the problem: a resolved inconsistency might not stay resolved. Frameworks in which inconsistency is tolerated help by allowing resolution to be delayed. However, evolution of a specification may affect both resolved and unresolved inconsistencies. We address these problems by explicitly recording relationships between partial specifications (ViewPoints), representing both resolved and unresolved inconsistencies. We assume that ViewPoints will often be inconsistent with one another, and we ensure that a complete work record is kept, detailing any inconsistencies that have been detected, and what actions, if any, have been taken to resolve them. The work record is then used to reason about the effects of subsequent changes to ViewPoints, without constraining the development process.

Introduct on

In an evolving specification, considerable development time and effort is spent handling recurrent inconsistencies. Such inconsistencies are particularly prevalent during requirements engineering, when conflicting and contradictory objectives are often required by different stakeholders. Tools and techniques for detecting and resolving inconsistencies only address part of the problem: they do not ensure that a resolution generated at a particular stage will apply at all subsequent stages of the process.

In this paper, we propose an approach for managing inconsistencies that arise during the development of multi-perspective specifications, by explicitly recording consistency relationships between partial specifications, and by representing both resolved and unresolved inconsistencies. We use the ViewPoints framework for multi-perspective software development as a vehicle for demonstrating our approach, and illustrate our techniques by working through an example drawn from the behavioural specification of a telephone.

Y e ew o nts Fra ewor

We base this work upon a framework for distributed software engineering, in which multiple perspectives are maintained separately as distributable objects, called ViewPoints. We will briefly describe the notion of a ViewPoint as it is used in this paper. [9] provides a fuller account of the framework, and [7] gives an introduction to the issues of inconsistency management.

A ViewPoint can be thought of as an 'actor', 'role', or 'knowledge source' in the development process, combined
A ViewPoi Tc as, ely as dculti-pe119gineering, in sx.m.00work

tion to besepar' in the d 0.79

inconsissepar' in the d'

ViewPoint templates, which together describe the set of notations provided by the method, and the rules by which they are used independently and together.

The notion of a viewpoint was first introduced as part of requirements specification methods such as Structured Analysis [22] and CORE [17], and more recently deployed for validating requirements [16], domain modelling [5] and service-oriented specification [12, 14]. In our framework, we use ViewPoints to organise multi-perspective software development in general, and to manage inconsistency.

Inconsistency Management

In our framework, there is no requirement for changes to one ViewPoint to be consistent with other ViewPoints [8]. Hence, inconsistencies are tolerated throughout the software development process. This contrasts with many existing support environments which enforce consistency, for example by disallowing changes to a specification that lead to inconsistencies.

We view strict enforcement of consistency throughout the requirements process as unnecessarily restrictive. Partly this view arises from a consideration of the distributed nature of software development: it may not always be possible to check that particular changes are consistent with work in progress at another site. Consistency enforcement can also stifle innovation, causing premature commitment and preventing exploration of alternatives [15]. Finally, development participants are likely to have conflicting views about many aspects of the requirements, and exploration of these conflicts are greatly facilitated by the ability to express the alternative views.

The ability to express and reason with inconsistent specifications during software development overcomes many of these problems. However, we assume that eventually a consistent specification will be required as the basis for an implementation[1]. We therefore focus on the management of inconsistencies, so that the specification process remains a coordinated effort. Consistency checking and resolution can be delayed until the appropriate point in the process. As there is no requirement for inconsistencies to be resolved as soon as they are discovered, consistency checking can be separated from resolution.

In order to manage inconsistencies, the relationships between ViewPoints need to be clearly defined. In general, the relationships arise from deploying the software development method. For example, if a method involves hierarchical decomposition of a particular type of diagram, then two diagrams that are hierarchically related should obey certain rules. Similarly, a method which provides several notations will specify how those notations inter-relate. Thus, the possible relationships between ViewPoints are determined by the method.

Consistency checking is performed by applying rules, defined by the method, which express the relationships that should hold between particular ViewPoints [21]. The rules define partial consistency relationships between the different representation schemes. This allows consistency to be checked incrementally between ViewPoints at particular stages rather than being enforced as a matter of course. A fine-grained process model in each ViewPoint provides guidance on when to apply a particular rule, and how resolution might be achieved if a rule is broken [20].

The need to tolerate inconsistency has been recognised in a variety of areas, including configuration management [23], programming [3], logical databases [10] and collaborative development [18]. In [7], we discuss how co-ordination between ViewPoints can be supported without requiring consistency to be maintained. A key problem is to support resolution of inconsistencies in an incremental fashion, so that resolutions are not lost when the ViewPoints continue to evolve. We now present a scenario to illustrate how this process is supported.

Scenario

Our scenario involves the behavioural specification of a telephone. We assume the existence of a method which allows such specifications to be partitioned into separate ViewPoints. We begin by outlining the salient features of the method, before introducing the scenario.

The Method

Our method uses state transition diagrams to specify the required behaviour of a device, in this case a telephone. The method permits the partitioning of a state transition diagram describing a single device into separate

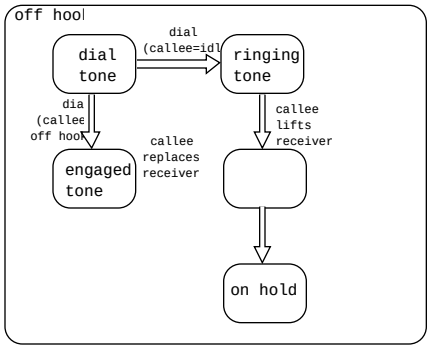
1 We will ignore the question of whether inconsistencies in a final specification or an eventual product are acceptable under some circumstances.

A owner Anne do a n te ephone ca ng

ew o nt A act ons
(1) delete trans t on off hoo_ d e
(2) move state connected so it is no longer part of state off hoo
(3) move trans t on off hoo_ d e so it no longer connects from state off hoo
(4) delete state connected
(5) delete state d e
(6) rename state connected
(7) rename state d e
(8) devolve trans t on off hoo_ d e to all sub-states of off hoo
ew o nt B act ons
(9) delete state connected
(10) delete state d e
(11) rename state connected
(12) rename state d e
Jo nt Act ons
(13) copy trans t on off hoo_ d e from ViewPoint A to ViewPoint B as trans t on connected d e

Table 1:

A owner Anne do a n te ephone ca ng



unknown action between the original resolution and the current action.

- The inconsistency re-appears, as is the case in our scenario. Here, the inconsistency is marked as unresolved, and annotated to show which actions resolved and re-introduced it. This allows ViewPoint owners to further eliminate suggested resolution actions, if they have been tried and found to be unsatisfactory.

D scuss on

Incremental exploration and resolution of the inconsistencies revealed an important mismatch between the conceptual models held by the two participants described in our scenario; namely about when connection are terminated, and whether there is a difference in being connected as a caller and connected as a callee. Although it is entirely possible that this mismatch may have been detected anyway, the explicit resolution process provides a