Feature Interactions in Aspect-Oriented Scenario Models

Gunter MUSSBACHER [a,1], Daniel AMYOT [a],
Thomas WEIGERT [b], and Thomas COTTENIER [c]
[a] *SITE, University of Ottawa, Canada*
[b] *Missouri University of Science and Technology, Rolla, MO, USA*
[c] *Hengsoft LLC, Palatine, IL, USA*

Abstract. Ideally, a feature interaction (FI) approach for scenario models should require minimal additional specification effort and allow for incremental definition of new scenarios. Furthermore, the scenario specification model and the validation model should be clearly separated. The specification model comprises the intrinsic behavioral and structural properties of the application under consideration, while the validation model consists of any additional information required for ensuring compliance with the feature specifications. The usage of pre- and postconditions in scenario models is one lightweight approach for detecting, modeling, and resolving FIs. It requires low effort, allows for incremental development, and if properly structured, allows for separation of the scenario and validation models. Aspect-oriented modeling techniques, however, add further requirements, as aspectual scenarios should remain independent from each other even if FIs are present. We present a specification and validation framework that satisfies the above-mentioned goals and assess its aspect-oriented scenario notation, Aspect-oriented Use Case Maps, against these goals based on a radio software case study.

Keywords. Feature Interaction, Aspect-Oriented Modeling, Scenario Models, User Requirements Notation, Use Case Maps

1. Introduction

Feature interactions (FIs) are commonly associated with the telecommunications domain [1] but are also of great relevance to many other domains and applications. In this paper, we focus on FIs in aspect-oriented scenario models for requirements where no detailed information about structural components is available and the modeler describes interactions between abstract components at a causal level without reference to message or data details. Once FIs have been detected, the modeler resolves them in the specification model. The validation model then ensures that the specification model remains compliant with the individual feature specifications and FI resolutions. This paper does not address the detection of FI, but assumes that FI have been discovered and discusses the combination of features, the resolution of FIs, and the testing thereof.

Ideally, it should be possible to define features and FI resolutions in an *incremental* way with *minimal effort*, they should be *easy to maintain*, the applied technique should *scale to large models*, the resulting specification and validation

---

[1] Corresponding Author: SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada; E-mail: gunterm@site.uottawa.ca.

models should be of *low complexity*, and the validation technique should be *effective* (i.e., be able to detect a large class of problems and interactions). Separation of concerns (SOC) is a key factor for all but the last of these goals. Aspects provide SOC and thus help with keeping individual features and even FI resolutions separate. If the specification model is organized in an aspect-oriented way, then the validation model also needs to reflect this organization to retain the encapsulation provided by aspects. Therefore, tests for features and FI resolutions need to be properly encapsulated, which in turn requires an aspect-oriented technique to be defined for the validation model. Furthermore, the specification and validation models themselves should be strictly separated from each other. This follows from good practice in software development that separates test code from production code.

Techniques based on pre- and postconditions generally require low effort compared to other formal techniques, but are still reasonably effective in detecting interactions. We present a specification and validation framework based on Aspect-oriented Use Case Maps (AoUCM) [2][3][4]. The specification model is a standard AoUCM model while the validation model is based on AoUCM's scenario definitions – a pre/postcondition technique. This paper does not focus on the detection of FIs but rather on problems discovered while modeling features and FI resolutions and defining the validation models. The suggested solutions to these problems may be generalized to other aspect-oriented scenario modeling techniques.

The remainder of this paper first provides background information on AoUCM including scenario definitions and gives a brief overview on related work in section 2. This paper's case study, a radio software system, is introduced in section 3 including its AoUCM specification and validation model. With the help of the case study, seven problems are identified and solutions for these problems described. Section 4 presents conclusions and a discussion on future work.

2. Background

2.1. Aspect-oriented Use Case Maps

Our specification and validation framework employs Aspect-oriented Use Case Maps (AoUCM) [3][4]. AoUCM is part of the *Aspect-oriented User Requirements Notation* (AoURN) [2], an approach that combines goal-oriented, scenario-based, and aspect-oriented modeling in one framework. The promise of aspects is that features and feature interactions can be more easily encapsulated in their own modules and selectively composed. In general, a UCM model consists of a path that begins at a *start point* (●) and ends with an *end point* (▌). A path may contain *responsibilities* (✗), identifying the steps in a scenario, and notational symbols for alternative (⤙) and concurrent (⊣⊢) branches. Path elements may be assigned to abstract *components* (□). For a more detailed description of UCMs, the reader is referred to [5].

In AoUCM, concerns and aspects are first-class modeling elements. An *aspect* encapsulates crosscutting concerns by grouping relevant *aspectual properties* as well as *pointcut expressions* needed to apply new scenario elements to a base model or to modify existing elements. An AoUCM *pointcut map* visually defines pointcut expressions, i.e. a pattern that must be matched for the aspect to be applied. For example, the pointcut map in Figure 1 matches against all maps in the base model that contain an OR-fork followed by a responsibility on at least one branch (see circled

portion of the base model). Grey start and end points are not included in the match but only denote the beginning and end of the pointcut expression. The aspectual properties are shown on a separate *aspect map* in AoUCM. The aspect map is linked to the pointcut expression with the help of a *pointcut stub* (✵).
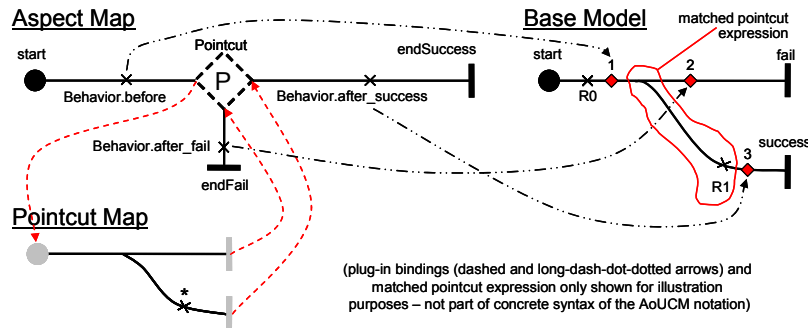

Figure 1. Aspect-oriented Use Case Maps

The causal relationship of the pointcut stub and the aspect's properties visually defines the composition rule for the aspect, indicating how the aspect is inserted in the base model (such as before/after/instead of/in parallel/interleaved or anything else that can be expressed with AoUCM). In Figure 1, the matched pointcut expression is indicated in the base model. The locations affected by the aspect are indicated by *aspect markers* (◆) which are inserted automatically by AoUCM's composition mechanism and are linked to the aspect map. For example, the aspectual behavior Behavior.before is added to the base model before the matched pointcut expression because it is located on the path before the pointcut stub. Behavior.before is therefore contained in aspect marker 1.

A modeler may further formalize UCMs with the help of standardized *scenario definitions* [5]. Each scenario specifies its pre- and postconditions as well as its start points and expected end points. For each choice point in the UCM model, Boolean expressions are defined for all alternatives. Boolean expressions may also be defined for start points, capturing preconditions of paths. The expressions may contain global Boolean, Integer, and Enumeration *scenario variables*. Responsibilities may change the value of variables. A scenario definition describes a specific path through the UCM model by initializing the variables used in the Boolean expressions and responsibilities. Given a scenario definition, the *traversal mechanism* [6] of the jUCMNav tool [7], the most comprehensive URN tool available, is capable of highlighting (executing) the scenario in a UCM model. With the help of the traversal mechanism, the scenario definitions effectively become a test suite for the UCM model.

UCM scenario definitions may be included in another scenario definition, which takes the union of the start points, end points, preconditions, and postconditions. The start points are ordered according to the include order of the scenario definitions, and initializations are applied to the variables also in the include order (i.e., an initialization of a scenario included later may override an initialization of a scenario included earlier).

2.2. Related Work

Blair and Pang [8] propose a two-level architecture that cleanly separates a feature's core behavior (in Java) from aspect-oriented code (in AspectJ), specifying how features

interactions are resolved. Our specification and validation framework is inspired by the general structure of this architecture but focuses on aspect-oriented modeling and not aspect-oriented programming.

There have been few attempts to handle aspect interactions during modeling. These approaches can be grouped into those that document interactions and those that detect interactions. Aspect interaction templates [9], precedences [9][10], and aspect interaction charts [11] explicitly document interactions. Kienzle et al. [12] define inter-aspect dependencies but do not consider conflicts. Formal methods (e.g., model checking [13], formal specifications of pre/postconditions [14], or static analysis [15]) have been applied to detect interactions. MATA [16] employs critical pair analysis to detect interactions and uses a numeric ordering scheme to capture precedence.

Feature interaction approaches are typically based on detecting structural interactions (e.g., [17]) or on applying formal methods such as model checking (e.g., [18]). UCMs have also been used in the past as a scenario-based approach for detecting and filtering feature interactions [19][20]. Other scenario-based approaches better emphasize the role of validation test suites, including the work on CRESS and MUSTARD [21]. However, these do not emphasize an aspectual decomposition, nor do they cope with the evolution of test scenarios in the validation model for handling composed features and feature interaction resolutions.

In contrast to our specification and validation framework, all of these approaches do not explicitly take into account how the validation model can be organized in an aspect-oriented way, how the validation model may be evolved as new features become available, and how validation and specification concerns are to remain separate.

3. Case Study: Radio Software

3.1. Introduction

The case study for this paper involves software that is responsible for controlling the correct functioning of a radio device, for example, a car radio. The radio device has eight buttons which can be pressed by the user. In total, ten features are supported by the radio software as detailed in Table 1. The table shows the name of the feature, the button or series of buttons by which the feature is invoked, and a short description of the feature. Alternative buttons are indicated by "/", a series of buttons is indicated by ",", and an arbitrary number of repeated buttons (at least one) is indicated by "*".

3.2. Aspects for Individual Features

A premise of aspect-oriented scenario modeling is that each scenario should be treated as fully independent, ignoring any potential for feature interactions. Each feature is modeled incrementally. Consequently, the UCM model follows the philosophy that each feature is ready to be executed at any time (as user interface restrictions should not be assumed). Because many features may be active at the same time, they may interfere with each other in an unwanted manner, or they may need to interfere to create the total behavior. These interactions need to be understood and resolved, whether the interaction is wanted or not. This paper does not address the detection of the interactions, but it deals with the proper encapsulation and modeling of features and feature combinations and the testing thereof.

Table 1. Radio Software Features

#	Feature Name	Button	Feature Description
1	Select Band	Band	The user switches between the AM and FM band.
2	Tune	FreqUp/Down	The user tunes the radio to a higher / lower frequency.
3	Autotune	Search	The user invokes automatic scanning for a frequency with a strong signal. When the end of the band is reached, the scan will wrap around.
4	Save	Memory*, Preset*, Memory	The user saves the current band and frequency.
5	Recall	Preset	The user recalls the saved band and frequency.
6	Traffic News	n/a	The radio prefaces traffic news with a distinctive signal.
7	Power On / Off	Power	The user can turn on the device, if it is turned off. The device must be on for any other feature to be enabled. The user can turn off the device, if it is turned on.
8	Standby On / Off	Standby	The user can set the standby mode, if the device is not in standby mode. Only a select number of features are available in standby mode. The user can cancel the standby mode, if the device is in standby mode.
9	Update Display	n/a	Whenever the frequency is changed, the radio's LCD display is updated with the new frequency, and the display (and the frequency) are held for a predetermined amount of time. Normal processing continues thereafter.
10	Remember Settings	n/a	The radio device remembers the last band and frequency settings when it is powered down and defaults to these settings upon powering on.

Therefore as a first step, the modeler describes each feature as an aspect with an individual UCM as shown in Figure 2. These aspects, however, do not crosscut each other and therefore do not need aspect-specific modeling features. They are modeled as aspects right from the start to acknowledge that potential interactions may appear at a later point and that these interactions are better encapsulated with an aspect-oriented approach. The UCM specification model is the result of this modeling effort. All features are specified with standard AoUCM in the UCM specification model. The validation model, on the other hand, requires new capabilities as shown in this section.

As a second step, the modeler creates scenario definitions for these features to ensure that they behave as expected – this is the UCM validation model. This requires the definition of global variables, the formalization of OR-fork conditions, and pseudo-code to be added to responsibilities when variable values need to be changed. For example, the condition of the [AM] branch of the Select Band feature is set to "Band == AM" while the [FM] branch is set to "Band == FM". The setBand(AM) responsibility sets the variable Band to AM, while setBand(FM) sets it to FM. The Select FM scenario definition of the Select Band feature is as follows:

Start Points: select (this is where the scenario starts)
Initialization: Band = AM (in order to switch to FM, the current band must be AM)
End Point: selected (the scenario visits this end point during the traversal)
Postcondition: Band == FM (must be satisfied at the end of the traversal)

In this case study, the goal of the second step is to achieve complete branch coverage for all features of the UCM specification model. While other coverage criteria could have been used, the specific employed methodology is irrelevant for the issues at hand as the fundamental structure of scenario definitions remains the same.

Therefore, one scenario definition each is created for Save, Recall, and Update Display. Two scenario definitions each are created for Select Band, Tune, Power, Standby, Traffic News, and Remember Settings. As the Autotune feature is more

complicated than other features, a total of four scenario definitions are created for it, covering the two cases where a frequency with a strong signal can or cannot be found. The two remaining scenarios cover situations where the Search button is pressed a second time: once after finding a strong signal and once during an unsuccessful scan.
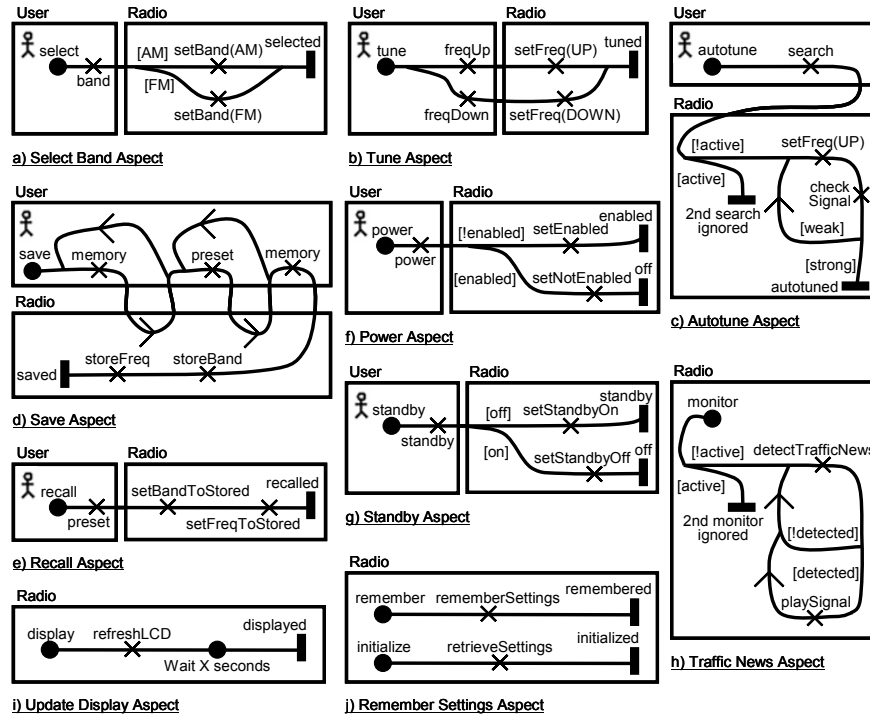

Figure 2. Aspects for Individual Features

Several problems were encountered during the formalization of the scenario definitions, i.e., when building the validation model. These are discussed in the following paragraphs and solutions are presented which were used for the UCM validation model in the case study.

Problem 1 – Scenario stops at path element that is not an end point: The Autotune feature in Figure 2.c is in an infinite loop if no strong signal is available. The traversal of this scenario therefore stops without reaching an end point once a predefined maximum number of iterations is reached. However, this is the expected behavior of this feature and therefore it should be possible to validate it as such.

Solution: Extend scenario definitions to allow model elements *other than end points* to be added to the list of expected end points (i.e., results). This extension leads to higher quality because a greater set of undesired modifications to the UCM specification model may be found based on the scenario definitions.

Problem 2 – Same feature several times in a row: In the Tune feature in Figure 2.c, the upper or lower branch is taken based on a variable called TuneDirection which can either be set to DOWN or UP. The modeler must initialize this variable in the scenario definition. While it is possible to add the same start point (e.g., tune) to the

scenario definition several times, a variable can only be initialized once in the scenario definition. Therefore, it is not easily feasible to define a validation scenario where the Tune feature is triggered several times in a row by the user with different values for the TuneDirection (e.g., UP, UP, DOWN, UP, DOWN, UP). A workaround solution, consisting of several variables (one for each time the feature is triggered) and a counter used in nested if statements, exists but does not scale and complicates the UCM specification model.

Solution: Expand the supported primitive types for scenario variables to *arrays*. If arrays are supported, a scenario variable can a) be initialized in the validation model by an array that consists of the desired values of the variable (e.g., TuneDirection = {UP, UP, DOWN, UP, DOWN, UP}) and b) be associated with a start point (e.g., tune). Each time the start point is triggered in the scenario (i.e., for each appearance in the scenario definition's list of start points), the next array cell is used to reinitialize the variable to a new value. The variable is then used normally in the UCM specification model (e.g., TuneDirection in the OR-fork of the Tune feature). Alternatively, the variable may also be associated with path elements other than start points (e.g., a responsibility). In this case, the next array cell is used each time the path element is visited by the path traversal. This expansion avoids unnecessary complications of the UCM specification model by a set of variables for the same user input. Furthermore, this also improves the separation of the UCM specification model and the validation model as this set of variables only exists for validation reasons and therefore pollutes the UCM specification model with validation-specific information,

Problem 3 – <feature>Active variables: The Autotune feature in Figure 2.c contains an OR-fork that checks whether the feature is active or not. Typically, this is accomplished by adding two responsibilities that set a variable such as autotuneActive to true just after the OR-fork and to false just before the end of the feature (i.e., the autotuned end point). This complicates the UCM specification model unnecessarily.

Solution: Make the *activity status* of a map during path traversal available to the modeler with a variable called _activeCount, thus allowing the traversal mechanism itself to determine whether a feature is active or not. _activeCount refers to the number of active traversals on a map (including sub-maps). Similarly, variables named _activeCount_<*mapName*> are used to reference other maps (i.e., features) from any map. With this approach the complexity of the UCM specification model is reduced. Only the branch conditions need to be specified for the OR-fork that checks whether a feature is active. For example, "_activeCount > 1" takes into account that there is for sure one active path on the feature's map. That is the path for which the traversal is currently visiting the OR-fork. If the feature is already active, then another active path exists and the value of _activeCount would therefore be greater than one.

Problem 4 – Control variables for loops: Often, the number of iterations has to be controlled for loops in a scenario. Sometimes it is possible and useful to introduce scenario variables to control a loop. For example, the variable Frequency is used in the loop of the Autotune feature in Figure 2.c. This is not an issue as the variable describes in a more formalized way what happens to the frequency in this and other features and therefore adds to the understanding of the UCM specification model (e.g., the frequency may go up or down). At other times, however, the introduced control variable is simply a counter. For example, the Save feature in Figure 2.d contains two loops because the Memory and Preset buttons may be pressed several times by the user. Two counter variables could be introduced here along with two responsibilities that increment the counters. This complicates the UCM specification model of the Save

feature with two responsibilities (e.g., _ctrLoop1 with code "_ctrLoop1 = _ctrLoop1 + 1;"). The branch conditions for the OR-forks will then use the counter variable (e.g., "_ctrLoop1 == NrMemoryPressed" and "_ctrLoop1 != NrMemoryPressed").

Solution: Make the *hit count* of a path element during path traversal available to the modeler with a variable called _hitCount. Variables named _hitCount_ <*pathElementID or responsibilityName*> allow referencing the hit counts of any path element from other path elements. With these variables, the validation model simply defines the maximum number of loop iterations (i.e., NrMemoryPressed) which is then compared to the hit count at the branching point. The counter variables as well as the responsibilities for incrementing the counters are not required anymore as this is now done by the traversal mechanism. The conditions for the OR-fork branches are now set to "_hitCount == NrMemoryPressed" and "_hitCount != NrMemoryPressed". Again, the UCM specification model is simpler and not polluted by validation-specific variables, keeping specification and validation concerns separated. Note that an array could also be used for the counter. This would require an array cell to be specified for each loop iteration. Consequently, arrays are better suited for values that change with each iteration, while hit counts are useful for capturing only the number of iterations.

3.3. Feature Interactions

The features defined in section 3.2 could actually operate the radio device, if no feature interactions (FIs) exist. FIs, however, have been identified for the radio features through a careful manual analysis of all feature combinations. These are explained in Table 2 which shows the feature that is currently executing in the Current column. The Next columns correspond to the next feature selected by the user. A filled cell indicates an interaction. An empty cell indicates that the next feature will execute after the current feature without interfering with the current feature. This is feasible since most of the radio features are short, run-to-completion-type features.

It is impossible (Imp. in Table 2) for Power On to occur when other features are already active because of the nature of the Power toggle switch. For the same reason, it is also not possible for Power Off to occur twice in a row. These interactions have already been resolved trivially in the Power feature itself with the variable called *enabled* and thus will not be discussed any further. Similarly, in the Standby feature, Standby On or Standby Off cannot occur twice in a row due to the Standby toggle switch. There are many ways in which the Power and Standby features could be combined. This case study assumes that the Standby feature can only be turned on or off, if the device itself is turned on. Therefore, Standby Off cannot occur if another feature is already active.

Table 2. Feature Interactions – Part I

#	Current	Next									
		Sel. B.	Tune	Autotune	Save	Recall	Tr. N.	P. On	P. Off	S. On	S. Off
1	Select Band							Imp.			Imp.
2	Tune							Imp.			Imp.
3	Autotune	Abort	Synergy	Ignore	Abort	Abort		Imp.	Abort	Abort	Imp.
4	Save	Abort	Abort	Abort		Ignore		Imp.	Abort	Abort	Imp.
5	Recall							Imp.			Imp.
6	Traffic N.						Ignore	Imp.	Abort		Imp.
7	Power On						Synergy	Imp.			
8	Power Off	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore		Imp.	Ignore	Ignore
9	Standby On	Ignore	Ignore	Ignore	Ignore	Ignore		Imp.	Abort	Imp.	
10	Standby Off							Imp.			Imp.

Three types of interactions occur: ignore, abort, and synergy. If the radio device is being powered off (or is already powered off), all features are ignored (except for the Power On feature which turns on the radio device). Standby On is very similar to Power Off in that all features are ignored except for the Traffic News, Power Off, and Standby Off features. If the device is in standby mode, the device can be turned off. Therefore, Power Off aborts Standby On (i.e., standby mode is cancelled).

The Save feature must be aborted if the button of any feature is pressed next except for Save's own Memory or Preset buttons. Save requires a series of interactions with the user, which could be interrupted if an unexpected button is pressed. Since the Preset button is also used by the Recall feature, the Recall feature could interfere with the Save feature and must therefore be ignored. The Traffic News feature does not require any buttons and runs in parallel to the Save feature without causing a feature interaction.

The Autotune feature may have to be aborted because it is a long-running feature. Therefore, the Autotune feature must be aborted if the button of any feature is pressed except for Autotune's own Search button which must be ignored and the TuneUp/Down buttons of the Tune feature as discussed below. The Traffic News feature again does not interact with the Autotune feature and therefore runs in parallel.

The Tune feature interacts with the Autotune feature as pressing the TuneUp/Down buttons may change the direction of the Autotune scan. This is an example of a synergetic interaction between features that creates something useful and new for the user. Now, the tune direction has to be considered by the Autotune feature. However, if the Tune feature is executed in the context of the Autotune feature, the Tune feature must not set the frequency as this may cause the Autotune feature to skip over a frequency with a potentially strong signal. Therefore, setting the frequency must be omitted in the execution of the Tune feature to avoid skipping over a frequency.

The Traffic News feature is a long-running feature that runs in parallel to all other features without causing interactions. It is only aborted by the Power Off feature. Note how Standby On aborts other features but not the Traffic News feature. If the Traffic News feature is executed twice, the second time is ignored. There is a synergetic effect between the Power On feature and the Traffic News feature, because the Traffic News feature must be started automatically by the Power On feature.

Furthermore, the Update Display and Remember Settings features interact with other features as indicated in Table 3. Update Display always occurs after the frequency is changed. Remember Settings occurs just before the radio device is powered down and just after the device is powered up. These interactions are examples of a simple combination of features that does not change each feature but defines when one feature must be executed in the context of another feature.

Table 3. Feature Interactions – Part II

#	Features	Sel. B.	Tune	Autotune	Save	Recall	Tr. N.	P. On	P. Off	S. On	S. Off
11	Update Display		ASF	ASF		ASF					
12	Remember Settings							ABF	BEF		

ASF…After setting frequency; ABF…After beginning of feature; BEF… Before end of feature

Therefore, a total of 33 interactions exist in this case study: 12 abort, 14 ignore, two synergy, three for Update Display, and two for Remember Settings. Note that the ignore interaction between Autotune and Autotune is not counted because it does not involve different features and therefore can be (and is) handled within the Autotune feature. The same applies to the ignore interaction between Traffic News and itself.

3.4. Aspects for Feature Interaction Resolutions

Aspects also model the FI resolutions. Standard AoUCM is sufficient for the specification of FI resolutions, but the validation model again requires new capabilities as discussed in section 3.6. Each aspect in section 3.4 addresses one or more of the 33 interactions identified earlier. Eight ignore interactions are dealt with the Power aspect (Figure 3.a). Its first pointcut expression matches against any user interaction except for pressing the Power button. Its second pointcut expression matches against the start point of the Traffic News feature. The aspect adds a check after each matched location, ensuring that the matched feature can only continue if the device is enabled (i.e., power is on). Otherwise, the execution of the feature is stopped by the aspect.

The UCMs in Figure 3.a and in Figure 2.f all belong to the same aspect. The base behavior of the Power feature is captured in Figure 2.f whereas the FI resolution is specified in Figure 3.a. The latter explicitly captures the decision on how to deal with the FIs of the Power feature in a separate UCM instead of distributing the resolution to all other features. This is one of the main advantages of an aspect-oriented approach for specifying FI resolutions and applies to all FI resolutions discussed in this section.
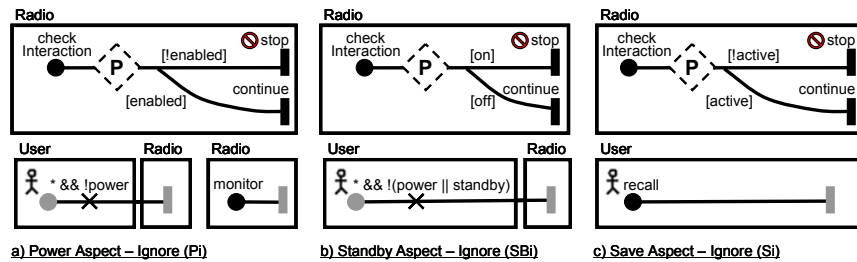

Figure 3. Ignore Interactions

a) Power Aspect – Ignore (Pi)   b) Standby Aspect – Ignore (SBi)   c) Save Aspect – Ignore (Si)

a) Autotune Aspect – Abort (Aa)   b) Save Aspect – Abort (Sa)   c) Traffic N. Asp. – Abort (TNa)   d) Standby Asp. – Abort (SBa)
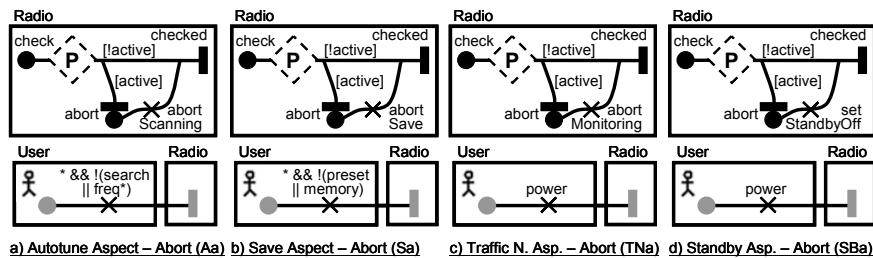
Figure 4. Abort Interactions

Five ignore interactions are dealt with the Standby aspect (Figure 3.b). Its pointcut expression matches against any user interaction except for pressing the Power or Standby buttons. As desired the Traffic News feature is not matched, because it does not require a user interaction. The aspect adds a check after each matched location, ensuring that the matched feature can only continue if the device is not in standby mode.

The last ignore interaction is dealt with the Save aspect (Figure 3.c). Its pointcut expression matches against the start of the Recall feature. The aspect adds a check there that disables the Recall feature if the Save feature is already active or allows the Recall feature to continue if the Save feature is not active.

The five abort interactions of the Autotune feature are dealt with the Autotune aspect (Figure 4.a). Its pointcut expression matches against any user interaction except for pressing the Search or FreqUp/Down buttons. The aspect adds a check after each matched location, ensuring that the Autotune scan is aborted if Autotune is active before the matched feature continues with its execution.

The five abort interactions of the Save feature are dealt with the Save aspect (Figure 4.b). Its pointcut expression matches against any user interaction except for pressing the Memory or Preset buttons. The aspect adds a check after each matched location, ensuring that the Save feature is aborted if it is active before the matched feature continues with its execution.

The abort interactions of the Traffic News and Standby features are dealt with the Traffic News aspect (Figure 4.c) and the Standby aspect (Figure 4.d), respectively. Their pointcut expressions match against the pressing of the Power button. The aspects each add a check after the matched location, ensuring that before powering off the device, the Traffic News feature is aborted if it is active and the Standby feature is cancelled (setStandbyOff) if it is active.
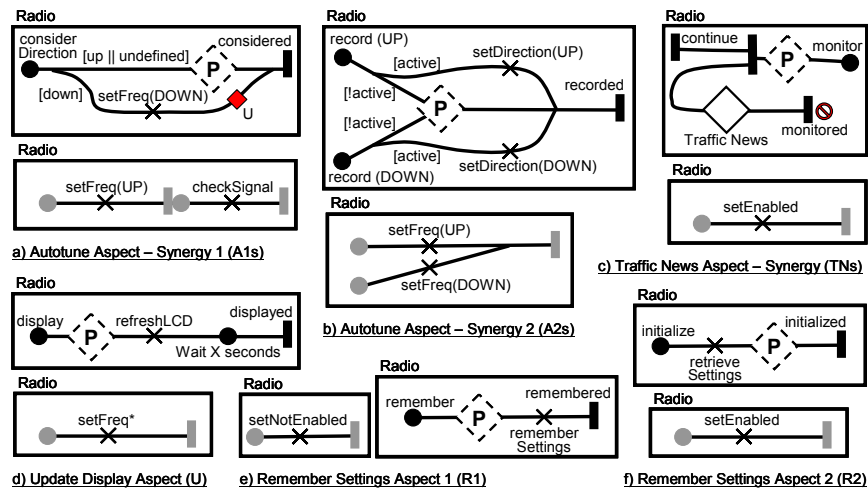

Figure 5. Synergy, Update Display, and Remember Settings Interactions

The synergy interaction between the Autotune and Tune features is resolved with the Autotune aspect (Figure 5.a and Figure 5.b). Two aspect maps with one pointcut expression each are required. The pointcut expression of the first aspect map matches against the setFreq(UP) and checkSignal responsibilities from the Autotune feature. The aspect then adds an OR-fork that checks the tune direction. If the tune direction is up or undefined, the aspect uses the already existing setFreq(UP) responsibility of Autotune. If the tune direction is down, the aspect replaces setFreq(UP) with setFreq(DOWN) (i.e., this is a conditional replacement as indicated by the conditional aspect marker (♠) in Figure 6.c). AoUCM allows a part of a matched expression to be replaced with the help of connected grey start and end points. By linking the pointcut stub to the connected end point and the start point on the far left, the modeler indicates that only this portion of the pointcut expression is to be changed.

The pointcut expression of the second aspect map matches against the two setFreq(UP) and setFreq(DOWN) responsibilities of the Tune feature followed by an

OR-join. The aspect then replaces the matched responsibilities with responsibilities that record the tune direction if the Autotune feature is currently active (i.e., this is also a conditional replacement, see Figure 6.b).

The synergy interaction between the Traffic News feature and the Power feature is dealt with the Traffic News aspect (Figure 5.c). Its pointcut expression matches against the setEnabled responsibility of the Power On feature. The aspect adds a parallel branch that starts the Traffic News feature after the setEnabled responsibility. The Traffic News stub links to the Traffic News map in Figure 2.h.

Finally, the five interactions for the Update Display and Remember Settings features are dealt with the aspects in Figure 5.d to Figure 5.f. These are straightforward aspects where a pointcut stub was added to each path in the Update Display and Remember Settings feature maps in Figure 2.i and Figure 2.j, respectively. The display is updated each time after the frequency has changed. The settings are remembered just after the device is disabled, and then initialized just before the device is enabled.

3.5. Complete AoUCM Model

The AoUCM model (Figure 6) shows all aspects applied to the UCM specification model. Each aspect marker is automatically added by the AoUCM composition mechanism and corresponds to exactly one of the 33 interactions from the two tables. The labels of the aspect markers match the abbreviations defined in Figure 3 to Figure 5.
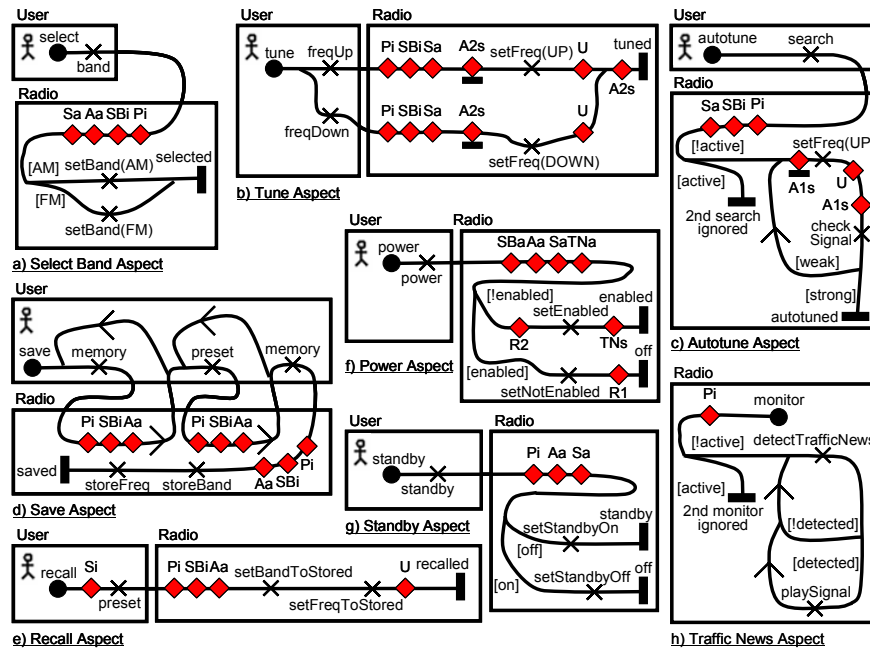

Figure 6. AoUCM Model

When several aspect markers are applied to the same location in the base model, there may be undesired interactions between these aspects. Precedence rules are typically used to resolve these interactions. A precedence rule defines the order in which two

aspects must be applied to the base model. In the case study, the Power aspect must have highest priority to ensure that no feature executes if the device is turned off. Standby has second-highest priority for similar reasons. Another interaction occurs between the Autotune-Synergy (A1s) aspect and the Update Display aspect in the Autotune feature (Figure 6.c). In this case, the Autotune aspect must have priority; otherwise the aspect marker for the Update Display aspect will end up after the second A1s aspect marker. If that happens the display will be refreshed twice, since an aspect marker for the Update Display aspect is already added in the A1s aspect (Figure 5.a). Note how the ordering of aspect markers reflects these priorities.

3.6. Scenario Definitions for Feature Interactions

As explained earlier, the modeler defines scenario definitions for each feature for validation purposes. Ideally, these definitions should be reused for the definition of interaction scenarios that correspond to FI resolutions, simply by including the scenario definition of an individual feature in the definition of an interaction scenario. However, this is generally not possible even for relatively benign interactions such as the interactions for Update Display and Remember Settings which only combine features but do not change them. This is due to the following problems encountered during the case study. The proposed solutions describe the required composition rules for an aspect-oriented validation model that covers features as well as FI resolutions. With these composition rules, the modeler can define all interaction scenario definitions. Essentially, an interaction scenario definition of a number of individual features must be able to reflect the changes to the scenario definitions of these features caused by FI resolutions. In other words, the modeler must be able to define composition rules that alter the interaction scenario definition in the desired way.

Problem 5 – Redundant start points: Each scenario definition of an individual feature must specify the start points of the feature, e.g., Power On starts at the power start point and Remember Settings starts at initialize. If the individual scenario definitions of these two features are included in an interaction scenario (see Power On column of row 12 of Table 3), the interaction scenario will contain the union of their start points (i.e., power and initialize). The execution of the interaction scenario by the traversal mechanism starts at the first start point (i.e., power on the Power AoUCM). The Remember Settings feature is fully traversed when its aspect marker R2 (Figure 6.f) is reached. The traversal of the Power feature then continues and finally comes to a stop when the enabled end point is reached. However, there is still the initialize start point in the list of start points to be triggered. Therefore, the traversal mechanism triggers this start point and traverses the Remember Settings aspect map (Figure 5.f) again. This is clearly not the desired behavior for the FI. This problem occurs for all interactions of the Update Display and Remember Settings features. However, the end points, initializations, and pre/postconditions of the two included scenario definitions should be merged as usual since they do typically describe the FI correctly.

Solution: Allow individual start points from an included scenario definition to be designated as *redundant*. Then, whenever a redundant start point is encountered by the traversal mechanism, it will not be triggered again. Alternatively, the modeler could declare the whole included scenario definition as redundant (i.e., redundant applies to all its start points), or even the whole interaction scenario definition (i.e., redundant applies to all start points of all included scenarios). Declaring a start point as redundant is not the same as removing the start point from the scenario definition in that the start

point is still expected to be visited by the traversal mechanism. The scenario definition for the Power On and Remember Settings interaction therefore is:

Interaction Scenario Definition = *include* Power On, Remember Settings;
start point initialize = *redundant*;

Problem 6 – Initializations, start/end points, and pre/postconditions may change: Problem 5 is exacerbated if more serious interactions occur that change the behavior of the involved features. In this case, start points may not be required anymore or have to be added, end points may not be reached anymore or other end points may now be reached, new initializations may be required, existing initializations may have to be overwritten, or pre/postconditions may change. For example consider the Power Off and Recall features and their interaction in the first four columns of Table 4. The initialization of the Power Off feature states that the device must be on to be turned off and the feature's postcondition states that the device must be turned off.

Table 4. Interaction Scenario Definition for Power Off and Recall

Scenario Definition	Power Off	Recall	Interaction	Improved Interaction
Included Scenarios	n/a	n/a	Power Off, Recall	Power Off, Recall
Start Points	power	recall	power, recall	power, recall
Initialization	enabled = true	n/a	enabled = true	enabled = true
End Points	off	recalled	off, recalled	off, ~~recalled~~, **stop**
Postcondition	!enabled	n/a	!enabled	!enabled

Improved Interaction Scenario Definition = *include* Power Off, Recall;
delete end point recalled;
add end point stop;

However, the Recall feature is ignored if the power is off. Therefore, the traversal ends at the stop end point of the Power aspect (Figure 3.a) instead of the recalled end point of the Recall feature because [!enabled] on the Power aspect map evaluates to true. The Power aspect was reached from the Power aspect marker Pi on the Recall AoUCM (Figure 6.e). However, only the continue end point on the Power aspect is connected to the out-path of the Power aspect marker, as indicated by the stop symbol (⊘) for the other end point of the Power aspect. The traversal therefore does not exit the Power aspect marker and the recall end point is never reached. Therefore, the execution of the interaction scenario definition by the traversal mechanism will fail.

Solution: Allow *deletions* of elements from included scenario definitions (see fifth column of Table 4). While additions (stop end point in Table 4 shown in bold) are already possible with current UCM scenario definitions [5], deletions are not (recalled end point in Table 4 shown crossed out and in bold). Note that it is also currently possible to override initializations. Additions, deletions, and overriding are appropriate mechanism for aspect-oriented scenario definitions. The original scenario definitions remain untouched and therefore unaware of any other feature or FIs. The modeler makes all changes in the interaction scenario definition which is aware of the features involved in the interaction. Thus, proper separation of concern among features, among FI resolutions, and between features and FI resolutions is upheld.

Problem 7 – Interrupting a scenario: The current semantics of scenario definitions allow the next scenario start point to be considered, only if the traversal has finished or is stuck somewhere in the scenario. This is rather limited as it does not allow, e.g., to create an interaction scenario that describes what happens if the user presses the Memory button (starts the Save feature) and then presses the Search button (aborts the Save feature, starts the Autotune feature). This is not possible, because once

the path traversal starts with the Save feature, the whole map (Figure 6.d) is traversed which includes the user pressing Preset and Memory again. The traversal cannot be interrupted before the preset responsibility to model the user pressing the Search button instead. Workaround solutions (involving UCM waiting places) exist but do not scale, complicate the UCM model, and pollute it with validation-specific information.

Solution: Enable scenario definitions to *interrupt* a scenario at a specific location in the scenario. An interruption defines a path element at which it occurs, whether it occurs before or after the path element, and a condition that must evaluate to true for the interruption to occur. For example, hit count variables may be used in the condition to indicate that the interruption occurs when the path element is visited the nth time. When the interruption occurs, the next start point in the scenario definition's list of start points is triggered. The modeler may use interruptions anywhere in the AoUCM model but they are particularly useful for loops and behaviors with multiple inputs from the user. See Table 5 for an example. The initializations for the Save feature ensure that the Memory and Preset buttons each will be pressed only once. The initializations of the Autotune feature ensures that the frequency will be increased by the setFreq(UP) responsibility starting at 32 until 51 is reached and the loop is exited.

Table 5. Interaction Scenario Definition for Save and Autotune

Scenario Definition	Save	Autotune	Interaction
Included Scenarios	n/a	n/a	Save, Autotune
Start Points	save	autotune	save, autotune
Initialization	NrMemory = 1, NrPreset = 1	Freq = 32, StrongFreq = 51	NrMemory = 1, NrPreset = 1, Freq = 32, StrongFreq = 51
End Points	saved	autotuned	~~saved~~, autotuned, **abort**
Postcondition	n/a	Freq == StrongFreq	Freq == StrongFreq

Interaction Scenario Definition = *include* Save, Autotune;
delete end point saved;
add end point abort;
interrupt before responsibility preset *if* _hitCount == 1

4. Conclusion and Future Work

We have presented an aspect-oriented technique for integrated specification and validation models at the requirements stage with which these models can be evolved to ensure compliance to feature specifications and FI resolutions. Seven problems that were hindrances in applying aspect-oriented techniques to the specification and especially validation models were discussed and solutions suggested. While the problems and solutions in section 3.2 may be specific to AoUCM, those in section 3.6 (i.e., the composition rules: include, add, delete, interrupt, redundant) are applicable in general to the validation of aspect-oriented scenario notations. The proposed framework contributes to the satisfaction of the ideal goals listed in the introduction, as the complexity of AoUCMs is reduced, their scalability is improved, feature concerns and FI resolution concerns are properly encapsulated, and the specification model is kept strictly separate from the validation model (thus further improving separation of concerns).

In future work, it may be possible to infer some of the composition rules for the validation model from the aspect-oriented composition of the specification model, thus further improving consistency. More internal information about the traversal mechanism in addition to hit counts and activity statuses may also be made available to

the modeler through special variables. Furthermore, an eighth problem was discovered during the case study. It addresses context definition for the validation model but was out of scope for this paper as it requires the introduction of concrete component instances, a rethinking of the global data model, and the introduction of scoping rules. Hence, the impact of supporting component instances on the AoUCM model, its semantics, and the validation framework requires further research and case studies.

References

[1] Bouma, L.G., Griffeth, N., and Kimbler, N. (Editors): "Feature Interactions in Telecommunications Systems". *Computer Networks 32:4*, 2000.

[2] Mussbacher, G.: "Aspect-Oriented User Requirements Notation". Giese, H. (Editor), Models in Sw. Eng.: Workshops and Symposia at MODELS 2007, Springer, LNCS 5002, pp 305-316, August 2008.

[3] Mussbacher, G., Amyot, D., and Weiss, M.: "Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps". Intl. Wksh. on Req. Eng. Visualization (REV 2006), Minneapolis, USA, 2006.

[4] Mussbacher, G., Amyot, D., and Weiss, M.: "Visualizing Early Aspects with Use Case Maps". Rashid, A. and Aksit, M. (Editors), TAOSD III, Springer, LNCS 4620, pp 105-143, 2007.

[5] ITU-T: *Recommendation Z.151 (11/08): User Requirements Notation (URN) – Language definition*, Geneva, Switzerland, 2008.

[6] Kealey, J.: *Enhanced Use Case Map Analysis and Transformation Tooling*. M.Sc. thesis, OCICS, University of Ottawa, Canada, 2007.

[7] *jUCMNav website*. University of Ottawa; http://jucmnav.softwareengineering.ca/jucmnav (acc. 01/09).

[8] Blair, L. and Pang, J.: "Aspect-Oriented Solutions to Feature Interaction Concerns using AspectJ". *Feature Interactions in Telecommunications and Software Systems VII*, IOS Press, pp 87-104, 2003.

[9] Sanen, F., Loughran, N., Rashid, A., Nedos, A., Jackson, A., Clarke, S., Truyen, E., and Joosen, W.: "Classifying and Documenting Aspect Interactions". *Workshop on Aspects, Components and Patterns for Infrastructure Software at AOSD*, Bonn, Germany, 2006.

[10] Zhang, J., Cottenier, T., Van den Berg, A., and Gray, J.: "Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver". *Journal of Object Technology 6*, pp. 89-108, 2007.

[11] Bakre, S. and Elrad, T.: "Scenario based resolution of aspect interactions with aspect interaction charts". *10th Intl. Workshop on Aspect Oriented Modeling (at AOSD)*, Vancouver, Canada, pp. 1-6, 2007.

[12] Kienzle, J., Klein, J., and Guelfi, N.: "RAM: Reusable Aspect Models - How to Specify Aspects in the Presence of Dependencies, Variabilities and Conflicts". Submitted to *TAOSD*. http://www.cs.mcgill.ca/~joerg/SEL/RAM_files/taosd2008_RAM_Kienzle.pdf (acc. January 2009)

[13] Shaker, P. and Peters, D.: "Design-Level Detection of Interactions in Aspect-Oriented Systems". *Aspects, Dependencies and Interactions Workshop at ECOOP 2006*, 2006.

[14] Mostefaoui, F. and Vachon, J.: "Design-level Detection of Interactions in Aspect-UML models using Alloy". *Journal of Object Technology 6*, pp 137-165, 2007.

[15] Douence, R., Fritz, T., Loriant, N., Menaud, J.-M., Segura-Devillechaise, M., and Südholt, M.: "An Expressive Aspect Language for System Applications with Arachne". *Aspect-Oriented Software Development (AOSD)*, Chicago, Illinois, 2005, ACM, 27-38.

[16] Jayaraman, P., Whittle, J., Elkhodary, A., and Gomaa, H.: "Model Composition in Product Lines and Feature Interaction Detection using Critical Pair Analysis". *MODELS 2007*, Nashville, TN, 2007.

[17] Liu, J., Batory, D., and Nedunuri, S.: "Modeling interactions in feature oriented systems". *Feature Interactions in Telecommunications and Software Systems VIII (ICFI)*, IOS Press, pp 178-197, 2005.

[18] du Bousquet, L.: "Feature Interaction Detection using Testing and Model Checking: Experience Report". *World Congress on Formal Methods in the Development of Computing Systems*, LNCS 1708, Springer, pp 622-641, 1999.

[19] Leelaprute, P., Nakamura, M., Matsumoto, K., and Kikuno, T.: "Design and Evaluation of Feature Interaction Filtering with Use Case Maps". NECTEC Technical Journal, pp 581-597, December 2005.

[20] Hassine, J.: *Feature Interaction Filtering and Detection with Use Case Maps and LOTOS*. M.Sc. thesis, University of Ottawa, Canada, February 2001.

[21] Turner, K.J.: "Validating feature-based specifications". *Software – Practice &. Experience 36:10*, pp 999-1027, August 2006.