7
Evaluation of the Architectural Metrics

The goal of this chapter is to evaluate the proposed concern-driven architecture metrics in terms of their usefulness to assess design modularity. It is also our goal to analyze to what extent concern-driven measurement is effective to cope with the limitations of conventional measurement approaches. To this end, we undertook three studies involving four systems (Sections 7.2, 7.3 and 7.4). We have used the concern-driven architectural metrics (Section 4.3) to perform modularity comparisons of two different architecture alternatives for each of the four target systems.

One of the alternatives is always an aspect-oriented (AO) architecture, while the second alternative is based on one or more specific architectural styles (Buschmann et al., 1996), herein referred as non-aspectual (non-AO) architecture. The AO architecture versions are also based on a hybrid composition of other conventional styles, which define the high-level design rules for the non-crosscutting concerns in the architecture design. In two of these studies (Sections 7.3 and 7.4), we also evaluated the usefulness of the metrics in order to quantify the impact of evolution changes in the modularity of the AO and non-AO architectures. The main fundamental reasons for consistently comparing AO and non-AO architectures are to:

- analyze to what extent the expected superiority of AO architectures in terms of separation of concerns can, in fact, be observed (or not) at the architectural stage based on our metrics suite;
- identify if architecturally-relevant crosscutting concerns, observed in typical architectural styles – such a publish-subscribe and layered architectures, can be effectively detected with our concern-driven metrics suite;
- detect the efficacy of our concern-sensitive metrics suite to detect inappropriate architectural decompositions using aspect-oriented mechanisms; and

- observe the positive and negative impacts that the obtained separation of concerns have in other equally-important modularity attributes in early design stages, such as architectural coupling, intra-component cohesion, and interface simplicity.

7.1. Study Procedures

This section describes some procedures that were executed to configure our empirical studies. The procedures presented here are common to the configuration of most of the studies. Procedures specific to each study will be presented in the preamble of the respective section describing the study. Before applying the metrics two major steps had to be done: (i) conception of architecture description whenever it was not available, and (ii) mapping of concerns to architecture elements. The second step was also executed in the study about detailed design metrics and heuristic rules (Chapter 8). In the following, we describe some procedures involved in these two activities.

Describing the architectures.

This step consisted of defining and describing the architecturally-relevant components, interfaces and operations that form the backbone of the architecture designs. Both AO and non-AO architectures in the three studies were conceived based on existing documentation and source code of the systems. First, we defined the components based on existing information about the design provided by the system's documentation. For instance, the documentation of the Health Watcher system contains a number of diagrams and natural language descriptions defining that the non-AO version of this system is structured around four layers: user interface, communication, business rules and data management (Soares et al., 2002). Therefore, we initially described the Health Watcher architecture consisting of four architectural components, each of them corresponding to a layer. After analyzing class diagrams and source code, we also included components for representing the transaction and concurrency control mechanisms.

In order to define the interfaces, operations and connections among them, we relied on class diagrams and source code. Based on these artifacts, we checked which classes and aspects were responsible to implement each component. Then, we searched for classes and aspects forming the boundaries of each architectural component, i.e. classes and aspects either used by or using classes of other components; and aspects affecting classes and aspects of other components. In order to define the interfaces and operations of an architectural component, we used the following criteria:

- Methods of classes or aspects in an architectural component *c* invoked by classes or aspects in an architectural component *c'* are described as operations of a provided interface of *c* and a required interface of *c'*. In addition these interfaces are connected to each other.
- Pieces of advice of an aspect within an architectural component *c* which are executed when methods of classes or aspects in an architectural component *c'* are described as operations of a provided interface of *c*. This provided interface is connected to the interface of *c'* which encompasses the operation representing the affected method. The connection is represented by means of an aspectual connector (Section 3.2).

Operations derived from methods or pieces of advice of the same class or aspect were usually grouped in the same interface. However, this is not a strict rule. Therefore, operations derived from different classes and aspects can be grouped in the same interface, if it makes sense. Besides, operations derived from the same class or aspect can be separated in distinct interfaces.

Assigning design elements to concerns.

The process of assigning design elements to concerns is critical to the success of the proposed measurement approach. In order to make the concern assignment task more systematic and facilitate the task of reliably deciding which design elements a concern should be assigned to, we followed a specific guideline in our empirical studies: assign a concern to a design element if the complete removal of the concern requires with certainty the removal or modification of the element. This guideline is inspired on the guidelines proposed by Eaddy et al (2007).

In addition, we undertook other procedures in order to support the concern-to-design mapping: (i) the activity of assigning design elements to concerns was done by two people assisting each other; we call this procedure as "pair mapping", and (ii) when possible, we consulted the actual developers of the systems whenever we were unsure about the mapping. We executed these procedures and followed the aforementioned guideline in the studies involving architectural metrics and also in the studies with design heuristic rules.

We are aware that variations in the assignment may imply variations in the measurement results. However, it is out of the scope of this work investigating approaches to support the designer on deciding which design elements are related to a concern. We plan to investigate this issue in the future (Section 9.2). Also, this should not influence much our key research questions as we need to primarily understand the extent that concern-based metrics and heuristics are useful compared with conventional metrics.

7.2. AspectT and MobiGrid study

The first study involved two systems, called AspectT (Garcia & Lucena, 2008; Garcia et al., 2004b, 2004c; Garcia, 2004) and MobiGrid (Barbosa & Goldman, 2004; Lobato et al., 2008). In this study, we performed a pair-wise comparison about the modularity of an aspect-oriented (AO) and a non-aspectual (non-AO) architectural solution of the two systems. The goal of this study is to evaluate how useful the concern-driven architectural metrics were in order to point out modularity-related differences between the two solutions. Thus, this study aims at verifying whether the metrics were able to show modularity-related improvements and drawbacks brought by the aspect-oriented solution in comparison to the non-AO one.

The assessment in this study included a subset of the metrics available in our current suite. It involved metrics for concern diffusion, concern-based cohesion, coupling between components (except Concern-Sensitive Coupling), and interface complexity (Section 4.3). The concern-sensitive coupling metric and the metrics for interaction between concerns were not used in this study as these metrics were defined based on the actual experience obtained in this first study.

One of the systems used in this first study, named AspectT, is an aspect-oriented agent framework (Garcia & Lucena, 2008; Garcia et al., 2004b, 2004c; Garcia, 2004) for implementing different kinds of software agents, such as, information and user agents. This framework have been used in the implementation of two instance applications: (i) Portalware – a Web-based system for the development of e-commerce portals (Garcia et al., 2004c), and (ii) ExpertCommittee – a multi-agent system (MAS) for conference management (Garcia et al., 2004b). Two versions of AspectT have been implemented using the AspectJ (Kiczales et al., 2001, The AspectJ Team, 2007) and Java programming languages.

The other system involved in this first study is called MobiGrid (Barbosa & Goldman, 2004; Lobato et al., 2008). It is a mobile agent system within a grid environment system. In this system, the agents can migrate whenever the local machine is requested by its user. Mobigrid has also two versions implemented in AspectJ and Java programming languages. Both versions encompass the usage of common multi-agent platforms and frameworks, such as JADE (Bellifemine et al., 1999), and Aglets (Lange & Mitsuru, 1998).

In the AspectT case, seven concerns were considered in the assessment process. Six are agent-specific behavioral properties, namely adaptation, interaction, autonomy, collaboration, mobility, and learning. The seventh concern, called kernel, consists of the core functionalities of a software agent. These concerns were chosen because these are the properties that should be reusable and easily (un)plugged from the software architecture. As in the MobiGrid system the architectural design was much more focused on modularising mobility-specific issues, the code mobility property and the MobiGrid application were the concerns considered in the assessment of this system's architectures.

In this context, both systems were ideal for our investigation due to several reasons. First, the chosen systems have stringent modularity requirements due to the demand for producing adaptable and evolvable architectures. Hence, all the system versions were developed with modularity principles as main driving design criteria, thereby motivating the exploitation of aspect-oriented software architectures. Second, the original non-AO architecture of each system was built in different contexts, laboratories and groups of developers (Garcia & Lucena, 2008; Barbosa & Goldman, 2004). Finally, they are systems that involve emphasis on uncommon crosscutting concerns, such as mobility, learning, autonomy, and their distinct compositions. As a consequence, there was no guarantee when the architectural choices were made that either the AO or non-AO architectural solutions were superior with respect to modularity requirements.

The non-AO architectural designs of AspectT and MobiGrid are presented in the following sections based on the UML 2.0 notation (OMG, 2005), while the aspect-oriented architectures are described based on AOGA (Section 3.2.1). Because of the use of the AOGA notation, we have to slightly adapt the definition of our architectural metrics. The adaptation was needed because our metrics take into account only provided and required interfaces (Section 4.2.1). AOGA includes an additional type of interface, namely crosscutting interface. As mentioned in Section 4.2.1, we consider only provided and required interfaces in definition of our metrics, because these are the most common types of interfaces in architecture description approaches, including aspect-oriented one. This makes our metrics more generic and easier to adapt to different architecture description languages.

The metrics adaptation merely consisted in treating crosscutting interfaces as provided interfaces in the metrics computation. This is because crosscutting interfaces plays a role very similar to the function played by provided interfaces: they are responsible for externalizing services provided by aspectual components. In fact, in some aspect-oriented architectural approaches, such as AO Visual Notation and AspectualACME (Section 3.2), the externalization of services provided by an aspectual component is done by a conventional provided interface. These approaches do not include a different type of interface for components playing the role of aspect.

Table 7 summarizes the configuration of the study presented in this section. Sections 7.2.1 and 7.2.2 focus on describing the main AO and non-AO architectural choices for both AspectT and MobiGrid systems. Section 7.2.3 presents measurement results, and Section 7.2.4 discusses them.

7.2.1.
The AspectT Architectures

Figure 46 shows the AspectT framework architecture described with the AOGA graphical notation (Section 3.2.1). It defines a set of aspectual components that address different agent concerns, such as interaction, autonomy, mobility, and learning. Crosscuts relationships (Section 3.2.1) connect aspectual components and the components they affect. Figure 47 shows a partial view of the operations in the interfaces of three components: Kernel, Interaction and Adaptation. Similarly to traditional interfaces in UML, a crosscutting interface is represented by a rectangle with an extra compartment in the bottom of it (Figure 47). This extra compartment represents the events and operations observed by the interface. The first compartment represents operations to be executed when an observed event is raised or observed operation is executed. This representation is inspired on the ASideML modeling language (Chavez & Lucena, 2001; Chavez et al., 2005). Following we describe the main components of the aspect-oriented architecture and their respective relationships.

Study goal	Analyze the usefulness of the metrics in order to point out modularity-related differences in the comparison of AO and conventional (non-AO) architectures of the same system.
Study Activities	1. Conception of the architecture descriptions, according to the procedure described in Section 7.1; 2. Mapping of architectural elements to the concerns, according to the procedure described in Section 7.1; 3. Metrics application (manually); 4. Analysis of the measurement results, comparing the modularity of the AO and non-AO architectures.
Target architectures	AO and non-AO architectures of AspectT and Mobigrid systems.
Arch. description approaches	UML 2.0 (OMG, 2005) for non-AO architectures; AOGA (Section 3.2.1) for AO architectures.
Considered concerns	AspectT: adaptation, interaction, autonomy, collaboration, mobility, and kernel; MobiGrid: mobility agent property, and MobiGrid application.

Used Metrics	Concern Diffusion over Architectural Components (CDAC);
	Concern Diffusion over Architectural Interfaces (CDAI);
	Concern Diffusion over Architectural Operations (CDAO);
	Lack of Concern-based Cohesion (LCC);
	Architectural Fan-in (AFI);
	Architectural Fan-out (AFO);
	Number of Interfaces (NI);
	Number of Operations (NO).

Metrics Adaptation	Crosscutting interfaces in AOGA are regarded as provided interfaces in the computation of the metrics

Table 7: AspectT and MobiGrid study configuration

The aspect-oriented architecture has the Kernel component as a central element. This component defines four interfaces: (i) KnowledgeUpdating – used to update the agent knowledge (belief, goal and plan), (ii) Services – which allows exposing the agent services, (iii) PlanExecution – which exposes events related to the execution of agents' plans, and (iv) LearningKnowledge – used to update the agent domain-specific knowledge based on machine learning techniques. KnowledgeUpdating is a provided interface which defines operations such as addBelief(), setGoal(), addPlan() and removePlan() (Figure 47). These operations are called by other components, such as Adaptation and Autonomy in order to update the agents' knowledge (Figure 46).

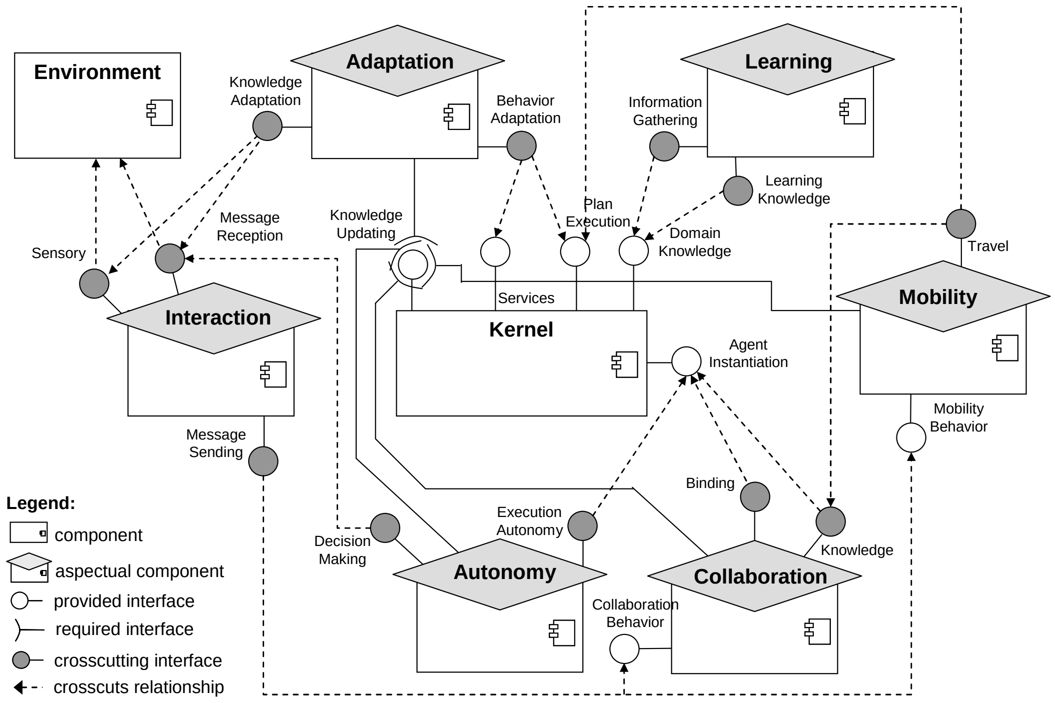

Figure 46: The AspectT architecture

A set of aspectual components are used to address different crosscutting agent concerns. Each of them either introduces new behavior in other components or refines the components' behavior by observing specific services' execution. The Interaction aspectual component is used to modularize the crosscutting impact on the use of communication architectures, such as JADE (Bellifemine et al., 1999). The Interaction aspectual component specifies crosscutting interfaces for message receiving (MessageReception) and for message sending (MessageSending). For instance, the MessageSending interface observes events raised by other components (outgoing message events) in order to know when it is time to send a message.

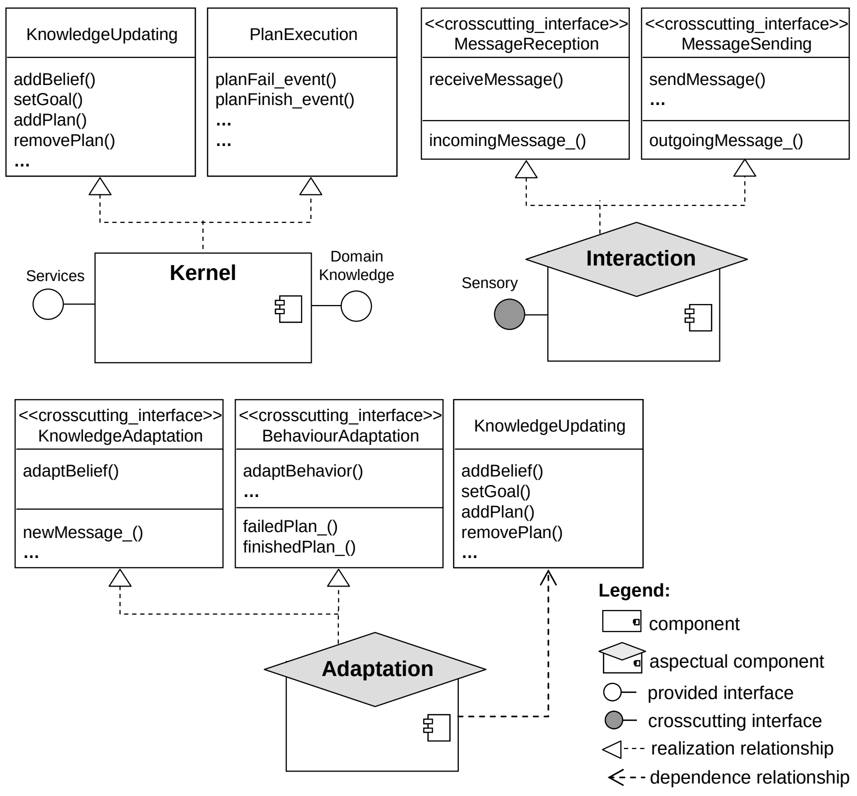

Figure 47: Details of AspectT interfaces

Messages are sent by means of the execution of the SendMessage() operation (Figure 47). For example, the MessageSending interface observes the execution of the Mobility component; when an agent is about to move from the current environment, the Mobility component raises an event by means of its MobilityBehavior interface. Then, the Interaction component detects this event and sends a message to the other agents informing them that an agent is leaving the environment.

The Adaptation component intercepts the MessageReception interface of the Interaction component by means of the KnowledgeAdaptation crosscutting interface. By means of the KnowledgeUpdating required interface, it updates the agent beliefs when new external messages are received. For instance, when a new message about a moving agent is received, the Adaptation component detects it. Then it uses the adaptBelief() operation in its KnowledgeUpdating interface (Figure 47) to require the Kernel component to adapt the other agents' belief, making them aware about the moving agent.

The Autonomy aspectual component defines two crosscutting interfaces: (i) DecisionMaking – affects the Interaction component to create new goals when new external messages are received, and (ii) ExecutionAutonomy – associates the agent with its own thread of control and also makes it possible the concurrent execution of agent plans. The Collaboration, Mobility and Learning components encompass crosscutting agent properties that are necessary only on specific agent architectures. The Collaboration component contains two crosscutting interfaces: (i) Knowledge – introduces new knowledge associated with roles to be played by the agent, and (ii) Binding – affects specific services from the Kernel component in order to instantiate new roles and attach them to the agent according to certain conditions. It also contains a provided interface – CollaborationBehavior – which raises collaboration events. The Interaction aspectual component observes these events in order to know when it is time to send a message.

The Mobility aspectual component is used to overcome the crosscutting nature of mobility concerns caused by the direct use of existing mobility platforms (Lange & Mitsuru, 1998). The Mobility component uses the Travel interface to introduce mobility capacities to the agent and to determine the execution points in which the agent can be moved. Finally, the Learning component is responsible for collecting information to execute its learning algorithms (InformationGathering interface). It also introduces new learning-specific knowledge associated with these algorithms (LearningKnowledge interface).

The AspectT framework has been developed as an alternative to an equivalent non-aspectual mediator-based architecture, presented in Figure 48. The latter defines a central component which mediates all the communication between the other ones. The Kernel component plays this central role. In our evaluation study involving AspectT, we used our metrics suite to assess the modularity of AspectT architecture (Figure 46) in comparison with its equivalent non-AO architecture (Figure 48)[5].

7.2.2. The MobiGrid Architectures

The original MobiGrid architecture ((Barbosa & Goldman, 2004) was defined based on the object-oriented mobility framework provided by the Aglets

[5] In both Figure 46 and Figure 48, each required interface is named the same as the provided interface it is connected to.

platform (Lange and Mitsuru, 1998), and follows a publisher-subscriber pattern (Buschmann et al., 1996). Lobato et al (2008) extended the original MobiGrid so as to make it flexible in the sense that not only Aglets, but also any other platform could be used to provide mobility capabilities to the agents. To this end, they built two versions of the MobiGrid system, an aspect-oriented (AO) and an object-oriented (which we call non-AO), which modularized and separate the mobility concern by means of different mechanisms. The former version used aspects and the latter used the publisher-subscriber pattern. In our evaluation study, we compared the modularity of the two versions.

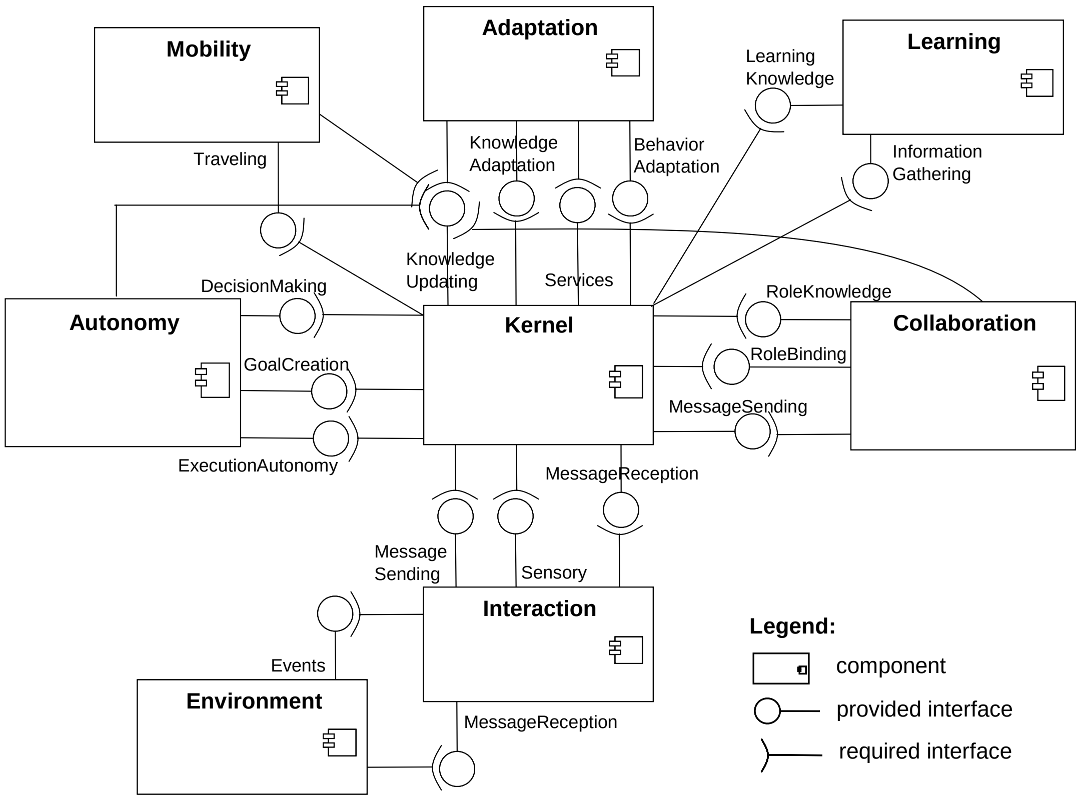

Figure 48: Non-AO mediator-based architecture equivalent to AspectT architecture

In both solutions, the separation of the mobility concern and the integration between MobiGrid and distinct mobility platforms respectively resulted in the conception of two architectural components: MobilityProtocol and MobilityManagement. Figure 49 illustrates the non-AO architecture, and Figure 50 presents the aspect-oriented one.

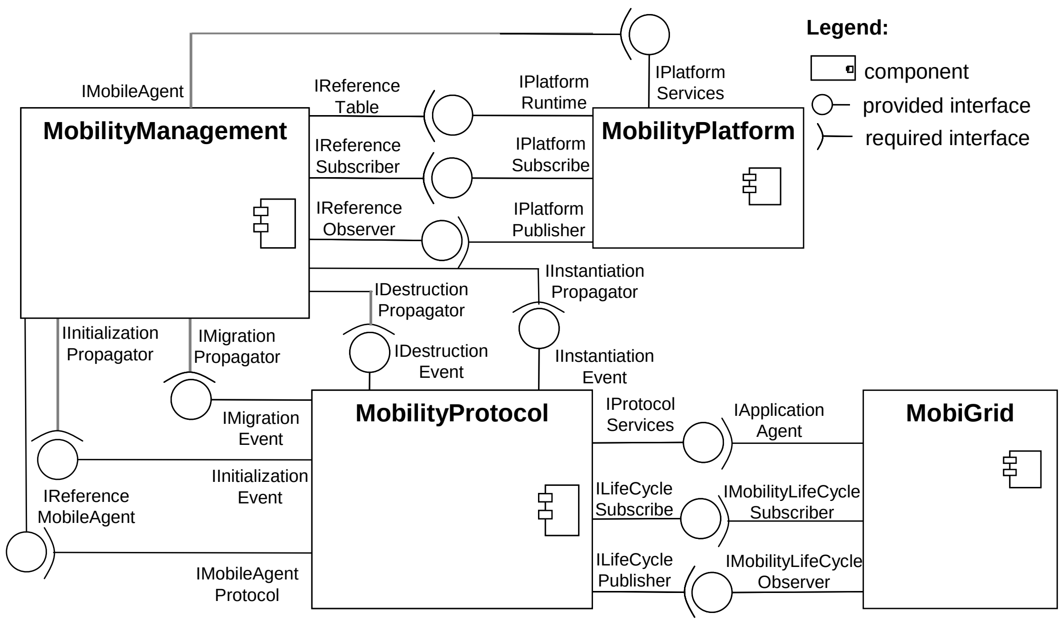

Figure 49: The non-AO MobiGrid architecture

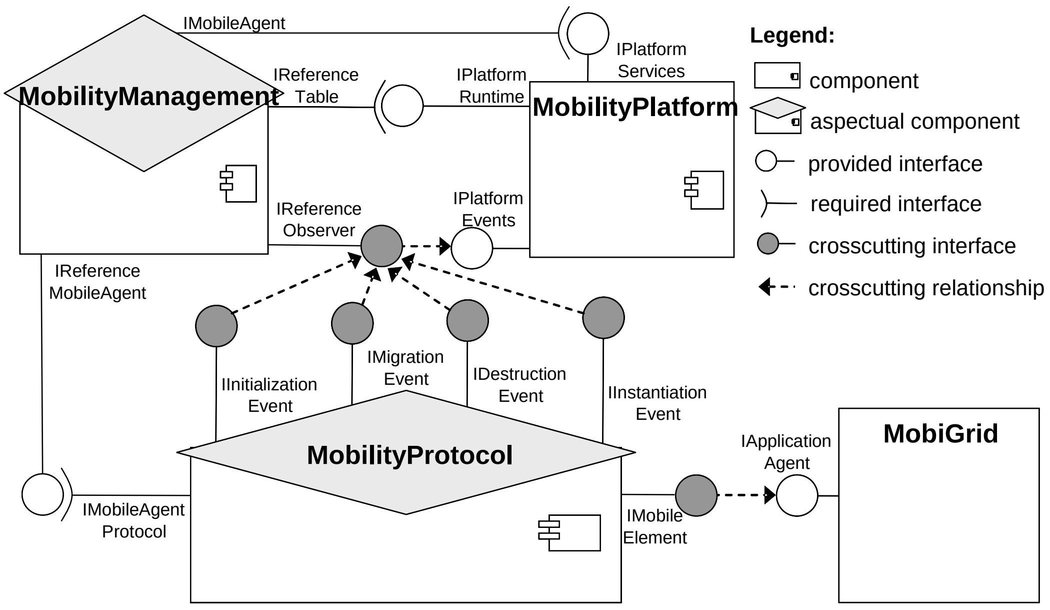

Figure 50: The AO MobiGrid architecture

In both cases, the MobiGrid architecture is composed of four kinds of components: (i) MobiGrid component, which modularises the basic concerns of an agent-based application, (ii) MobilityProtocol component, which modularises the mobility protocol execution – i.e., the instantiation, migration, remote initialisation, and destruction of MobiGrid agents, (iii) MobilityManagement component, which provides a flexible integration between MobiGrid and distinct mobility platforms, and (iv) MobilityPlatform, which represents a specific mobility platform being used, such as Aglets (Lange and Mitsuru, 1998) or JADE (Bellifemine et al., 1999).

In both architectures, the main purpose of the MobilityProtocol component is the explicit separation of the mobility concerns from the MobiGrid component. In addition, the MobilityManagement component connects the MobiGrid with the MobilityPlatform component, which modularises and externalises the platform services.

The AO architecture in Figure 50 uses the crosscutting interface abstraction (Section 3.2.1) to make it possible a clean modularization of the mobility concern in the MobiGrid system. The MobilityProtocol component now implements a generic mobility protocol in order to prevent the explicit invocations of the mobility services by the MobiGrid component. Such explicit invocations happen in the non-AO architecture due to the interaction constraints imposed by the publisher-subscriber pattern. Therefore, we invert the way in which access to the mobility services is typically designed in mobile agent systems.

To do that, the IMobileElement crosscutting interface is used to determine when and how a mobile agent is instantiated on a platform to represent a specific agent on the MobiGrid. This interface also triggers the agent migration to other environments, since the mobile agent may have to migrate whenever elements of the MobiGrid are called or executed. That is, the IMobileElement interface is used to affect well-defined mobility join points in order to determine when MobiGrid agents should move. Thus, the IMobileElement interface allows an explicit separation of mobility issues from the other MobiGrid non-crosscutting concerns.

7.2.3. Results

This section presents the results of the measurement process involving the AspectT and MobiGrid architectures. The results presentation is broken in three parts. The first part presents the results for the concern diffusion metrics. The second part presents the results for the coupling and cohesion metrics. Finally, the last part presents the results for the interface complexity metrics. The results are shown by means of tables that put side-by-side the values of the metrics for the

AO and non-AO architectures of each system. For all the employed metrics, a lower value implies a better result.

Concern Diffusion

In the quantitative evaluation of the AspectT framework, the data collected for both AO and non-AO architectures show favorable results for the AO version for most of the metrics. Table 8 presents the complete data collected for the AspectT architectures considering the concern diffusion metrics. These metrics count the total number of components, interfaces and operations dedicated to realize a concern (Section 4.3.1).

Concerns	Concern Diffusion over Architectural Components (CDAC)		Concern Diffusion over Architectural Interfaces (CDAI)		Concern Diffusion over Architectural Operations (CDAO)	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
Interaction	2	1	9	3	22	10
Adaptation	2	1	6	2	34	5
Autonomy	2	1	7	3	80	31
Collaboration	2	1	6	4	87	37
Mobility	2	1	3	3	35	20
Learning	2	1	4	2	16	6
Kernel	1	1	2	4	14	68

Table 8: AspectT: concern diffusion measures

We can observe differences between the AO and non-AO versions for all the concern diffusion metrics. Table 8 shows that the non-AO architecture, which follows the mediator pattern (Gamma et al., 1995), requires two components to address each of the system concerns (CDAC metric), except for the kernel concern. It happens because the Kernel component needs to inevitably embody functionalities from the different concerns in addition to kernel-specific functionalities. It occurs because the Kernel component plays the mediator role and, as a consequence, propagates information relative to every concern to the ‘colleague’ components. For example, besides the Interaction component, the interaction concern is also present in the Kernel component; the latter realizes an interface, named MessageSending (Figure 48), exclusively dedicated to cooperate with the Interaction component in order to send messages to the agent's surrounding environment.

On the other hand, the design of the Kernel component and its interfaces are not affected by other concerns in the AO architecture. This occurs because the Kernel component does not require or pass any information to the other components. It only exposes events related to its basic functionalities which are observed by the aspectual components affecting it. For instance, the Adaptation aspectual component observes the execution of plans by means of the aspectual connection between its BehaviourAdaptation crosscutting interface and Kernel's PlanExecution provided interface. When a plan is executed, Adaptation obtains information about the executed plan and updates the agent knowledge.

We can also observe in Table 8 that the AO version requires fewer interfaces (CDAI metric) and operations (CDAO metric) for most of the system concerns with exception of the kernel concern. The kernel concern in the AO version is represented by the Kernel component. This component needs to expose new interfaces in the AO version to enable the aspectual components to observe and get information about events related to its functionalities. However, all these additional interfaces are part of the kernel concern and, therefore, separation of architectural concerns is not hindered.

Table 9 shows the results for the concern diffusion metrics for both MobiGrid architecture options. Again, the AO architecture performed better than the non-AO one, which follows the publisher-subscriber pattern. As shown in Table 9, the mobility concern is scattered over fewer architectural components in the AO architecture (CDAC metric). This concern is present in four components in the non-AO architecture, whereas it crosscuts only three components in the AO architecture. This occurs because, in the non-AO architecture, the MobiGrid component encompasses two mobility-related interfaces – IMobilityLifeCycleObserver and IMobilityLifeCycleSubscriber – for explicitly processing of mobility life cycle events. These events are captured by the IMobileElement crosscutting interface in the AO architecture. Although this difference does not seem to be significant, the AO architecture makes the mobility-related interfaces unnecessary in the MobiGrid component.

Concerns	Concern Diffusion over Architectural Components (CDAC)		Concern Diffusion over Architectural Interfaces (CDAI)		Concern Diffusion over Architectural Operations (CDAO)	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
Mobility	4	3	23	13	407	326
Application (MobiGrid)	1	1	1	1	18	18

Table 9: MobiGrid architectures: concern diffusion measures

The concern diffusion metrics also showed better results for the AO architecture in terms of number of interfaces (CDAI metric) – 13 versus 32 – and number of operations (CDAO metric) – 326 versus 407. This is mainly caused because the MobilityProtocol and MobilityManagement aspectual components need fewer interfaces and operations for handling events. This will be further discussed in Section 7.2.4.

Architectural coupling and concern-based cohesion

Table 10 and Table 11 present the measurement for architectural coupling and concern-based cohesion metrics considering, respectively, the AspectT and MobiGrid architectures. The tables in this subsection also put side-by-side the metrics values for the AO and non-AO architectures. However, as the values here are per component (component point of view), the bottom of the tables also provides the total values (sum of all the component measures) that represent the results for the overall architecture point of view. The table bottom presents the tally of all the component measures: the rows labeled 'Total' indicate the tally for the system architecture, while rows labeled 'Diff' indicate the difference (in percentage) between the AO and non-AO architectures for the system point of view relative to each metric. A positive value means that the non-AO architecture fared better, whereas a negative value indicates that the AO architecture exhibited better results.

As we can observe in Table 10, there is an expressive coupling increase in the non-AO AspectT architecture considering the number of requiring components (Architectural Fan-in metric). The fan-in is 12 in the mediator-based architecture, while it is nine in the AO architecture, representing a difference of 25% in favor of the latter. This occurs because the services of several aspectual components in the AO version (e.g., Adaptation, Autonomy, and Learning) are not requested by other components. This phenomenon is granted to the dependency inversion promoted by AO architectures.

Components	**Lack of Concern-based Cohesion (LCC)**		**Architectural Fan-out (AFO)**		**Architectural Fan-in (AFI)**	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
Kernel	7	1	6	0	5	5
Interaction	1	1	2	3	2	2
Adaptation	1	1	1	2	1	0
Autonomy	1	1	1	2	1	0
Collaboration	1	1	1	1	1	1
Mobility	1	1	1	2	1	1
Learning	1	1	0	1	1	0
Total	**13**	**7**	**12**	**11**	**12**	**9**
Diff	**–46.2%**		**–8.3%**		**–25.0%**	

Table 10: AspectT architectures: coupling and cohesion measures

Components	**Lack of Concern-based Cohesion (LCC)**		**Architectural Fan-out (AFO)**		**Architectural Fan-in (AFI)**	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
Mobility Platform	1	1	1	0	1	1
Mobility Manager	1	1	2	1	2	1
Mobility Protocol	1	1	2	2	2	0
MobiGrid	2	1	1	0	1	1
Total	**5**	**4**	**6**	**3**	**6**	**3**
Diff	**–20.0%**		**–50.0%**		**–50.0%**	

Table 11: MobiGrid architectures: coupling and cohesion measures

As stated in Section 4.3.3, we assess the lack of cohesion of a component counting the number of distinct concerns addressed by it, which is captured by the Lack of Concern-based Cohesion (LCC) metric. LCC measurement resulted in better results for the AO version (13 versus 7 = 46.2%). This superiority is justified by the fact that in the mediator-based architecture the Kernel component, besides realizing the kernel concern, needs to implement required interfaces associated with the other six concerns. Hence, there is an explicit architectural tangling in the Kernel component.

The AO architecture of the MobiGrid system presented better outcomes in terms of the two coupling metrics and in terms of the cohesion metric as well (Table 11). The non-AO architecture exhibited architectural fan-out 50% higher than the AO architecture. This difference is a consequence of the reduction of fan-out in both MobiGrid and MobilityManagement components in the AO version, since they do not have to explicitly call the MobilityProtocol component for notifying events. Being an aspectual component, MobilityProtocol captures the events by means of crosscutting interfaces.

MobilityPlatform also contributes for decreasing the fan-out, because it does not need to be connected to the MobilityManagement component in order to notify events. In this case, the aspectual MobilityManagement component observes the events by means of its IReferenceObserver crosscutting interface. For the same reasons, the architectural fan-in metric also showed worse results for the publisher-subscriber version of the architecture (50% higher). In this case the fan-in reduction is observed in the MobilityProtocol and MobilityManagement components.

Interface Complexity

Table 12 and Table 13 show the results for the interface complexity metrics for the AspectT and MobiGrid architectures, respectively. Regarding the AspectT system (Table 12), the metrics demonstrate the modularity benefits obtained in the AO version compared to the non-AO one. There was a bigger difference in the number of interfaces specified for each version (37 versus 22 = 40.5%) which favors the AO version. This difference is mainly due to the additional interfaces of the Kernel component, but it is also thanks to the values collected for other components. The increase in the number of interfaces metric for the mediator version is also reflected in the number of operations. Table 12 shows that the number of operations is 38.5% higher in the non-AO version. Again, it happens because the Kernel component plays the mediator role and, as a consequence, it has additional interfaces and operations to propagate information relative to every concern to the other ‘colleague’ components.

Components	Number of Interfaces (NI)		Number of Operations (NO)	
	non-AO	*AO*	*non-AO*	*AO*
Kernel	16	4	115	68
Interaction	5	3	13	10
Adaptation	4	3	29	5
Autonomy	4	3	49	31
Collaboration	4	4	47	37
Mobility	2	3	19	20
Learning	2	2	16	6
Total	**37**	**22**	**288**	**177**
Diff	**–40.5%**		**–38.5%**	

Table 12: AspectT architectures: interface complexity measures

Components	Number of Interfaces (NI)		Number of Operations (NO)	
	non-AO	*AO*	*non-AO*	*AO*
Mobility Platform	4	3	185	176
Mobility Manager	9	4	155	124
Mobility Protocol	8	6	61	26
MobiGrid	3	1	24	18
Total	**24**	**14**	**425**	**344**
Diff	**–41.7%**		**–19.1%**	

Table 13: MobiGrid architectures: interface complexity measures

The use of aspects had a strong positive influence in the interface complexity of the MobiGrid architectural components, as shown in Table 13. For the non-AO architecture, the number of interfaces was more than 40% higher than in the AO solution. Also, the number of operations was higher in the non-AO solution (19.1%). The main reason for this result is the decrease on the number of required interfaces of the MobilityManagement aspect. In the non-AO solution, the conventional component has four required interfaces to propagate four mobility events relative to the initialization, migration, destruction and instantiation of agents. These events are observed by the IReferenceObserver interface and propagated to the MobilityProtocol component.

On the other hand, in the AO solution, the aspectual component MobilityProtocol affects the IReferenceObserver interface and directly observes the events when MobilityPlatform notifies them. Hence, the required interfaces to propagate them are not necessary. Moreover, the inferiority of the non-AO version in the number of interfaces is granted to the fact that it needs additional pairs of subscription interfaces involving the collaboration of the components MobilityManagement and MobilityPlatform, and the components MobilityProtocol and MobiGrid components.

7.2.4. Discussion

This section provides a more general analysis with respect to the results previously presented and discusses how the metrics were useful to point out modularity anomalies. The use of the architectural metrics allowed us to observe: (a) modularity-related differences in the investigated architectures, (b) the manifestation of certain crosscutting concerns in the architectural stage, and (c) when AO architectures are well designed, they can also affect positively other equally-important modularity attributes in addition to separation of concerns. Moreover, the results show that the joint use of concern-driven and conventional metrics is a promising approach to improve architecture modularity assessment. We also discuss some limitations observed in the study as well. In fact, the issues discussed here inspired us to define new metrics such as Concern-Sensitive Coupling (Section 4.3.4) and Number of Concern Interfaces (Section 4.3.5), which are used in the next study. Our observations are classified into three main categories: (a) crosscutting concerns, (b) bidirectional architectural coupling, and (c) architectural interface bloat.

Crosscutting concerns

A crosscutting concern is a concern scattered over multiple design modules and tangled with other concerns (Tarr et al., 1999). The concern diffusion measures supported the identification of scattered concerns. For instance, the results of the Concern Diffusion over Architectural Components metric showed that the interaction property is scattered over two components in the non-AO AspectT architecture (Table 8). In addition, the results of the Lack of Concern-based Cohesion metric showed that the Kernel component embodies seven concerns in the non-AO AspectT architecture.

Checking the concerns addressed by the Kernel component, we see that one of these concerns is the interaction property. Therefore, the interaction property may be considered as a crosscutting concern, since it is spread over multiples components and tangled with other concerns. Thus, the joint use of Concern Diffusion over Architectural Components and Lack of Concern-based Cohesion metrics seems promising to identify crosscutting concerns. Nevertheless, it still requires checking the components with more than one concern to identify the concerns tangled in each component. This limitation motivates us to define metrics for quantifying interaction between concerns (Section 4.3.2), which are used in the study presented in the next section.

Bidirectional architectural coupling

After a careful joint analysis of the MobiGrid and AspectT architectures, we observed that both non-AO options – i.e., the mediator-based and the publisher-subscriber designs – imposed some undesirable bidirectional couplings. In the mediator architecture, all the ‘colleague’ components need to inevitably have references to the ‘mediator’ component and vice-versa. Similarly, in publisher-subscriber architecture, all the ‘subscriber’ components need to know the ‘publisher’ components and vice-versa.

Even though these architectural solutions overcome the problem of direct couplings between colleagues and between subscribers, the AO architectural solutions for both MobiGrid and AspectT systems have reduced even more the overall architecture couplings by making almost all the inter-component relationships unidirectional (aspectual components affect the components). For example, the Kernel component has the fan-out zero in the AO version of the AspectT architecture, against six in the non-AO version (Table 10). The use of aspectual components removed the agent properties (interaction, adaptation, etc.) from the Kernel component and, as a consequence, removed the coupling imposed by these concerns to the component.

This phenomenon is observed mostly from the Architectural Fan-in and Architectural Fan-out measures (Table 10 and Table 11), which showed the decrease on the degree of coupling, and the Lack of Concern-based Cohesion metric, which showed the decrease on the number of concerns in the Kernel component. However, grasping that the decrease of coupling is related to the decrease on the number of concerns is not straightforward. In order to reduce this limitation, we proposed the Concern-Sensitive Coupling metric (Section 4.3.4), which quantifies the contribution of each concern to the coupling of a component.

Architectural interface bloat

The inter-component interaction constraints defined by the mediator-based and publisher-subscriber architectures did not scale respectively in the AspectT and MobiGrid systems, causing a complexity increase in the component interfaces. Such constraints have influenced the definition of extra operations and additional interfaces for the sake of realizing certain agent concerns, such as mobility and learning issues. For example, the evidence of interface bloat can be observed in several parts of both non-AO architectures. As discussed in Section 7.2.3, the Kernel component in the AspectT design (Table 12) and the MobiGrid component (Table 13) had clearly much 'wider boundaries' respectively due to their needs of mediating inter-component conversations and handling event subscriptions and notifications.

In the particular case of MobiGrid, the event propagation is an issue that crosscuts the modularity of all the four architectural components. Again, the joint analysis of the results of interface complexity metrics and the Lack of Concern-based Cohesion metric showed that the realization of certain concerns imposed an increase on the interface size of certain components. Based on this observation, we proposed the Number of Concern Interfaces metric (Section 4.3.5), which quantifies the contribution of each concern to the number of interfaces of a component.

7.3. Health Watcher study

This study involved a typical Web-based information system, called Health Watcher, and consisted of a pair-wise comparison about the modularity of AO and non-AO architectural designs of the system. Similarly to the study with AspectT and MobiGrid (Section 7.2), the goal of this study is to evaluate the usefulness of the concern-driven architectural metrics in order to find out modularity-related differences between the two solutions. However, in the assessment of Health

Watcher architectures, we included new metrics not used in the previous study, namely the concern-sensitive coupling metric and metrics for interaction between concerns.

This study also included a step where we used the metrics for analyzing the modularity of the Health Watcher architectures in the context of evolutionary scenarios. A series of changes was undertaken in the implementation of both AO and non-AO versions of Health Watcher (Greenwood, 2007a), and we applied the metrics before and after these changes. The goal of this step was to analyze how the metrics performed in order to assess the impact of evolution changes in the architecture modularity. We further describe the procedures of this step in the last part of Section 7.3.2.

Health Watcher is a Web-based information system that supports the registration and management of complaints to the public health system. This system was selected because it met a number of relevant criteria for our intended evaluation. First, it is a real system with existing Java and AspectJ implementations (each around 4000 lines of code). The first Health Watcher release of the Java implementation was deployed in 2001 by the Public Health System in Recife, a city located in the north of Brazil (Soares et al, 2002). Since then, a number of incremental and perfective changes have been addressed in posterior Health Watcher releases.

Second, this system has been served as a kind of benchmark for the assessment of contemporary modularization techniques, such as AOSD (Filho et al., 2007; Greenwood et al., 2007a; Pinto et al., 2007; Kulesza et al., 2006; Sampaio et al., 2007; Silva et al., 2007; Soares et al., 2002). In addition, the Health Watcher was suggested as the target system for case studies in the submissions to the Early Aspects at ICSE: Workshop in Aspect-Oriented Requirements Engineering and Architecture Design (Early Aspects at ICSE, 2007). Third, both object-oriented and aspect-oriented designs of the Health Watcher system were developed with modularity-driven requirements, such as reusability and maintainability, as main driving design criteria.

The concerns considered in the measurement process of this study were concerns typically found in information systems: graphical user interface (GUI), distribution, business rules, concurrency, persistence and exception handling. Table 14 summarizes the configuration of the study presented in this section.

Section 7.3.1 focuses on describing the main AO and non-AO architectural choices for Health Watcher system. The architectures are described based on UML 2.0 (OMG, 2005) and AO Visual Notation (Section 3.2.2). The metrics definitions are compliant with the abstractions introduced by these two architecture specification approaches, thus it was not necessary to adapt them. Section 7.2.3 presents measurement results, and Section 7.2.4 discusses them.

Study goals	Analyze the usefulness of the metrics in order to: (i) point out modularity-related differences in the comparison of AO and conventional (non-AO) architectures of the same system, and (ii) assess the differences on the impact of evolution changes in the modularity of both architecture versions.
Study Activities	1. Conception of the architecture descriptions, according to the procedure described in Section 7.1; 2. Mapping of architectural elements to the concerns, according to the procedure described in Section 7.1; 3. Metrics application (manually); 4. Analysis of the measurement results, comparing the modularity of the AO and non-AO architectures; 5. Implementation of six change scenarios in both AO and non-AO versions; 6. Update of the architecture description in order to reflect the changes; 7. Update the concern-to-design mapping; 8. Application of the metrics (manually); 9. Analysis of the measurement results, assessing the impact of changes in the architecture modularity.
Target architectures	AO and non-AO architectures of the Health Watcher system.
Arch. description approaches	UML 2.0 (OMG, 2005) for the non-AO architecture; AO Visual Notation (Section 3.2.2) for the AO architecture.
Considered concerns	Graphical user interface (GUI), distribution, business, concurrency, persistence and exception handling.

Used Metrics	Concern Diffusion over Architectural Components (CDAC);
	Concern Diffusion over Architectural Interfaces (CDAI);
	Concern Diffusion over Architectural Operations (CDAO);
	Component-level Interlacing Between Concerns (CIBC);
	Interface-level Interlacing Between Concerns (IIBC);
	Operation-level Overlapping Between Concerns (OOBC)
	Lack of Concern-based Cohesion (LCC);
	Concern-Sensitive Coupling (CSC);
	Number of Interfaces (NI);
	Number of Operations (NO).

Metrics Adaptation	Not necessary

Table 14: Health Watcher study configuration

7.3.1. The Health Watcher Architectures

Figure 51 illustrates a graphical representation of non-AO architecture of the Health Watcher system based on UML 2.0 notation (OMG, 2005). Figure 52 presents the aspect-oriented version of the Health Watcher architecture based on the AO Visual Notation (Section 3.2.2). Both the non-AO and AO architectural designs are mainly determined by the conjunctive application of both client-server and layered architectural styles (Buschmann et al, 1996).

Six main architectural concerns were considered in the Health Watcher architectural design: graphical user interface (GUI), distribution, business, persistence, concurrency and exception handling. The first four are represented by layers in the non-AO solution (Figure 51); however the distribution concern has been aspectized and is no longer a layer in the AO version (Figure 52). Table 15 briefly explains the Health Watcher's architecture elements correlating them to the concern they address. Figure 51 and Figure 52 also show how the concerns are spread over the architecture as already described in Section 6.3.2.

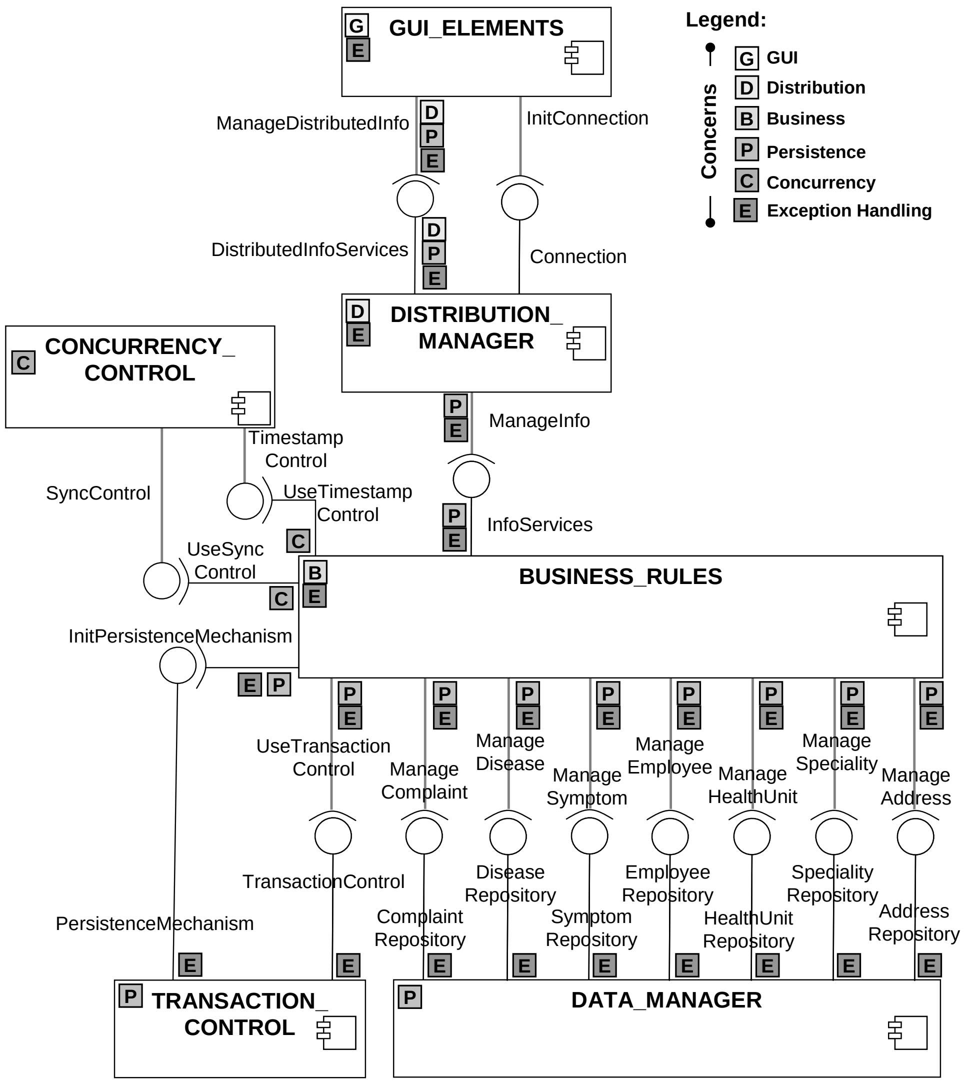

Figure 51: Non-AO architecture of the Health Watcher system

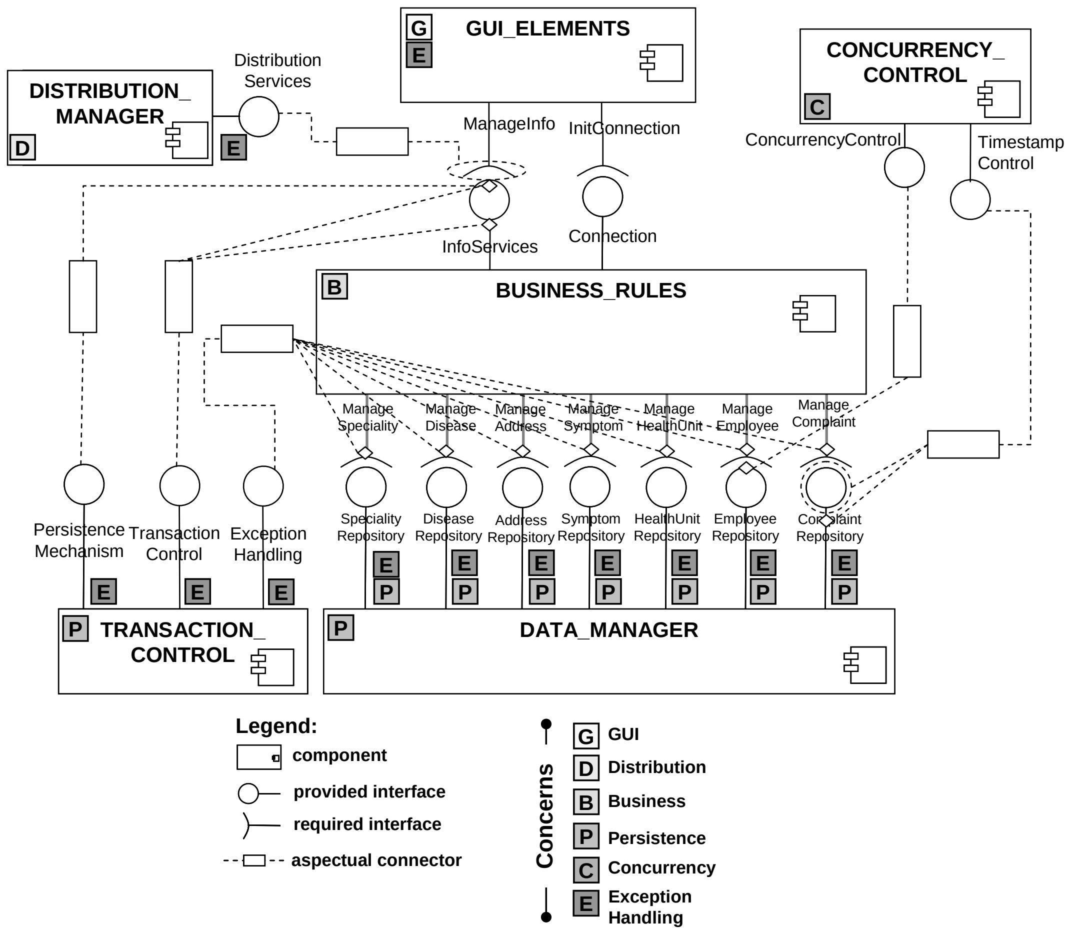

Figure 52: Aspect-oriented architecture of the Health Watcher system

Concern	Description
GUI	The GUI_Elements component provides a Web interface for the system.
Distribution	The Distribution_Manager component externalizes the system services at the server side and support their distribution to the clients
Business	The Business_Rules component defines the business elements and rules
Persistence	The Data_Manager and Transaction_Control components address the persistency concern by storing the information manipulated by the system and providing transaction control, respectively.
Concurrency	The Concurrency_Control component provides control for avoiding inconsistency in the information manipulated by the system
Exception Handling	Exceptional events are raised and handled by the components.

Table 15: Health Watcher architectural concerns

In the Health Watcher system, complaints are registered, updated and queried through a Web client represented by the GUI_Elements component. These services are provided by the Business_Rules component by means of the InfoServices interface. In the non-AO version, the GUI_Elements component accesses the Business_Rules' services through the Distribution_Manager component. This component allows client (GUI_Elements) and server (Business_Rules) to be distributed in different computers and supports the remote communication between them. In the AO version, Distribution_Manager is an aspectual component, which makes the remote access to the Business_Rules component transparent to GUI_Elements.

The Data_Manager and Transaction_Control components address the persistence concern which includes data storage, connection and transaction control. In the non-AO version, the Business_Rules component invokes the Transaction_Control's services, such as begin transaction and commit transaction, when a data-related operation is executed in its InfoServices interface. In the AO solution, Transaction_Control is an aspectual component which affects the InfoServices interface and executes the transaction control services when an operation is executed in that interface.

The concurrency concern is addressed by the Concurrency_Control component which, similarly to Transaction_Control, is invoked by the Business_Rules component in the non-AO architecture, and affects its interfaces in the AO solution. The Concurrency_Control deals with different facets of concurrency, including the timestamp. Timestamp is a technique used to avoid object inconsistency. This problem can occur when two copies of an object are retrieved by different requests before one of them can update its version. The technique uses a timestamp field to avoid object updating if there is a newer version of it stored in the persistence mechanism.

7.3.2. Results and Discussion

This section presents the results of the measurement process involving the Health Watcher architectures and discusses how the concern-driven metrics tackle the limitations of conventional architecture metrics. The results presentation is broken in five parts. The first four parts present the results about the comparison between the AO and non-AO architectures and discuss how they support overcoming the limitations of conventional metrics (Section 1.2). The last part describes the application of the metrics in different releases of both versions of the architecture (evolution scenarios). Most of the results are shown by means of tables that put side-by-side the values of the metrics for the AO and non-AO architectures of each system.

Identification of non-localized concerns

Table 16 presents the measures for concern diffusion relative to the AO and non-AO versions of the Health Watcher architectures. The results show that most of the concerns are spread over more architecture elements in the non-AO solution. For instance, in the non-AO architecture, the persistence concern affects more components (CDAC metric) – 5 vs. 2, more interfaces (CDAI metric) – 22 vs. 10 – and more operations (CDAO metric) – 154 vs. 46. This occurs mainly because in the AO solution the persistence-specific exceptional events are modularized within the Transaction_Control aspectual component and, as a consequence, do not need to be addressed by the interfaces of Business_Rules, Distribution_Manager and GUI_Elements components as in the non-AO solution.

Concerns	**Concern Diffusion over Architectural Components (CDAC)**		**Concern Diffusion over Architectural Interfaces (CDAI)**		**Concern Diffusion over Architectural Operations (CDAO)**	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
GUI	1	1	2	2	14	14
Distribution	2	1	4	1	51	16
Business	1	1	8	9	57	57
Persistence	5	2	22	10	154	46
Concurrency	2	1	4	2	8	4
Exception Handling	5	4	22	11	156	52

Table 16: Health Watcher: concern diffusion measures

The exception handling concern is scattered over five components in the non-AO architecture against four in the AO version (CDAC metric). Although this difference does not seem to be significant, the difference in terms of interfaces and operations is much higher in favor of the AO solution: 22 vs. 11

interfaces (CDAI), and 154 vs. 46 operations (CDAO). This happens because the only component from where the exception handling concern is totally removed in the AO solution is the Business_Rules. However, this is the component with the higher number of interfaces and operations, and almost all of them are affected by the exception handling concern in the non-AO solution.

Similar situation occurs in the case of the distribution concern. The difference in favor of the AO architecture is only high in terms of interfaces and operations. This reason for that is because the Distribution_Manager component needs three interfaces in the non-AO solution and only one in the AO version. Moreover, a number of operations in the ManageDistributedInfo interface (GUI_Elements component) deals with distribution-related exception, which is no longer necessary in the AO architecture.

Identification of dependencies between architectural concerns

Table 17 presents the outcomes for interaction between concerns (CIBC, IIBC and OOBC metrics). The results show that modularizing some concerns with aspectual components decreases the interlacing between concerns at the component level in the AO architecture. For instance, note that the business concern is interlaced with three concerns at the component level (CIBC metric) in the non-AO architecture against none in the AO version.

Concerns	**Component-level Interlacing Between Concerns (CIBC)**		**Interface-level Interlacing Between Concerns (IIBC)**		**Operation-level Overlapping Between Concerns (OOBC)**	
	non-AO	*AO*	*non-AO*	*AO*	*non-AO*	*AO*
GUI	3	1	3	0	0	0
Distribution	3	1	3	1	1	1
Business	3	0	2	0	0	0
Persistence	5	1	4	1	1	1
Concurrency	3	0	0	0	0	0
Exception Handling	5	2	4	2	2	2

Table 17: Health Watcher: interaction between concerns measures

Checking the architecture description (Figure 51) and concern templates (Section 6.3), we can see that one of these three concerns is the persistence concern. One of the causes of this interlacing is the TransactionControl required interface (Figure 51), which is related to the persistence concern, in the

Business_Rules component. This interface is not necessary in the AO architecture, because the Transaction_Control aspectual component provides the transaction control service by capturing the requisitions to the data-related services directly in the InfoServices provided interface (Figure 52).

The CIBC results for the business concern (Table 17) complement the results provided by the concern diffusion metrics (Table 16). The latter say that the business concern is localized in only one component in both versions. The former say that, although well localized, the business concern interacts with three other concerns in the non-AO architecture (Table 17). This occurs because the Business_Rules component, which is mainly responsible for addressing the business concern, has some parts (interfaces and operations) affected by these other three concerns (persistence, concurrency and exception handling) in the non-AO architecture.

The interaction related to interface-level interlacing is also lower in the AO solution (Table 17). For instance, the persistence concern is interlaced with four other concerns at the interface level in the non-AO architecture, against only one concern in the AO solution (IIBC metric). This is due the fact that persistence-specific exceptional events are spread over interfaces of the Business_Rules, Distribution_Manager and GUI_Elements components in the non-AO architecture. On the other hand, in the AO solution, these events are handled by the Transaction_Control aspectual component which captures them directly in the provided interfaces of the Data_Manager component (Figure 52).

Finally, the results regarding the metric for operation-level overlapping (OOBC) show that the AO solution for the Health Watcher architecture was not able to reduce this kind of concern interaction (Table 17). In both AO and non-AO architectures the exception handling concern are overlapped with the persistence and distribution concerns due to the exceptional events specific to these two concerns. Even though these two concerns are modularized within aspectual components, the interfaces of these components still have to include the exceptional events.

Identification of unstable components

The results for the Lack of Concern-based Cohesion metric (LCC) (Table 18), which counts the number of concerns per component, says that there are four concerns in the Business_Rules component in the non-AO architecture against only one in the AO solution. Therefore, this component is more instable in the non-AO architecture in the sense that it is subject to the influence of more concerns in this version of the architecture. Changes relative to business, persistence, concurrency and exception handling may cause changes in the Business_Rules component. Note that unlike the results for the metrics in the previous tables of this section, the results for the metrics on Table 18 are gathered per component.

Components	Lack of Concern-based Cohesion (LCC)	
	non-AO	*AO*
GUI_Elements	4	2
Distribution_Manager	3	2
Concurrency_Control	1	1
Business_Rules	4	1
Transaction_Control	2	2
Data_Manager	2	2

Table 18: Health Watcher: concern-based cohesion measures

The architectural fan-out metric (AFO) (Section 4.3.6) is a conventional metric which quantify a dominant attribute, namely coupling. Fan-out metrics are usually used to quantify the instability of a component (Martin, 1997). Its value for the Business_Rules component (AFO = 3) would suggest that this component is one of the most unstable in the non-AO architecture, since it is coupled to three other components: Transaction_Control, Concurrency_Control and Data_Manager. Changes in these components would ripple effects to the Business_Rules. However, this metric is not able to suggest that the GUI_Elements is also one of the most unstable components in the non-AO architecture. This component is coupled to only one component (Distribution_Manager); nevertheless it may be changed because of changes relative to three concerns besides the GUI concern: persistence, distribution and exception handling. This information can be obtained by means of the Lack of Concern-based Cohesion metric.

Breaking the tyranny of the dominant architectural modularity attributes

Table 19 presents the results for the Number of Interfaces (NI) and Number of Operations (NO) metrics. Analyzing the results for these conventional metrics, we can see that two components, namely Distribution_Manager and Business_Rules, have more complex interfaces in the non-AO architecture. For instance, the Business_Rules component has more interfaces (12 vs. 9) and more operations (66 vs. 57) in the non-AO architecture. GUI_Elements has the same number of interfaces, but it has a slightly higher number of operations.

Components	**Number of Interfaces (NI)**		**Number of Operations (NO)**	
	non-AO	*AO*	*non-AO*	*AO*
GUI_Elements	2	2	**17**	**14**
Distribution_Manager	**3**	**1**	**34**	**16**
Concurrency_Control	2	2	4	4
Business_Rules	**12**	**9**	**66**	**57**
Transaction_Control	2	3	5	6
Data_Manager	7	7	40	40
Total	**28**	**24**	**166**	**137**
Diff	**–14.3%**		**–17.5**	

Table 19: Health Watcher: interface complexity measures

Although this information is important, concentrating the analysis only on interface complexity attribute does not give us any clue about the reasons for that difference. In this way, the results for the Lack of Concern-based Cohesion metric (LCC) (Table 18) can complement this information in the sense that it shows that those components which have more complex interfaces also have more concerns affecting them in the non-AO solution. Therefore, these concerns can be one of the causes for the higher interface complexity.

Nevertheless, the joint use of the metrics Lack of Concern-based Cohesion, Number of Interfaces and Number of Operations still limits the modularity analysis: it requires verifying in the architecture whether the difference in interface complexity is really due to the difference in number of concerns. This limitation can be overcome by using the Number of Concern Interfaces metric (NCI) (Section 4.3.5), as shown in Figure 53.

Figure 53 presents the results of the Number of Concern Interfaces and Concern-Sensitive Coupling metrics for the Business_Rules component in both versions of the architecture. Each bar shows how each concern contributes to the total value of the metric. The numbers in the bars represent the absolute value for each concern. It is important to remember that the Number of Concern Interfaces metric does not distingue between provided and required interfaces; and the Concern-Sensitive Coupling metric takes into account only the fan-out type of coupling, i.e., the coupling related to the "uses" relationship (Section 4.3.4)

The results for Number of Concern Interfaces metric clearly show that one third of interfaces in Business_Rules are dedicated to other concerns (persistence and concurrency) apart from the business concern in the non-AO architecture (Figure 53 – first bar). This may indicate that removing persistence and concurrency from this component would reduce the number of interfaces in 33%. The number of interfaces was, in fact, reduced from 12 to 9 (25%) in the AO version of the architecture (Figure 53), because an additional interface related to the business concern was required in the this solution.

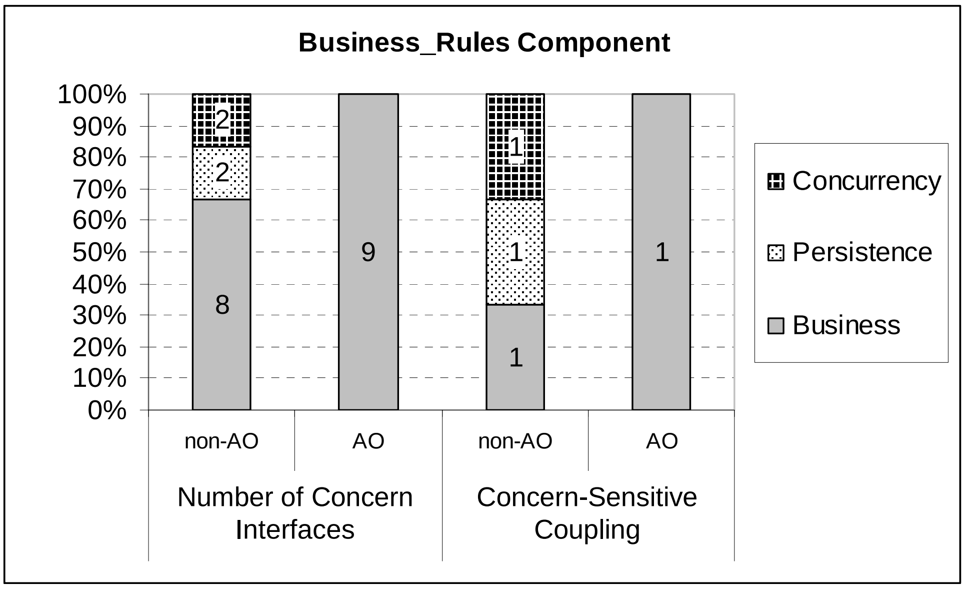

Figure 53: Business_Rules component: Number of Concern Interfaces and Concern-Sensitive Coupling measures

Similar analysis can be done for the Concern-Sensitive Coupling metric. As shown in Figure 53, concurrency and persistence concerns contributes to two thirds of the coupling (fan-out) of the Business_Rules component in the non-AO solution. This means that removing these concerns from the component, the coupling might be reduced in the same proportion. In fact, these concerns no longer exist in the Business_Rules component in the AO solution and, as a result, the coupling (fan-out) is reduced from 3 two 1. This kind of analysis cannot be done based only on conventional coupling metrics.

Applying the metrics in the context of evolution scenarios

We present here the results of using the metrics in the context of an evolution scenario involving the Health Watcher system. This phase of the study involved the implementation of a series of changes in both Health Watcher versions (available from Greenwood et al. 2007c). The selected changes applied to the Health Watcher system vary in terms of the types of modifications performed. Some of them add new functionality, some improve or replace functionality, and others improve the system structure for better reuse or maintainability. The purpose was to expose the non-AO and AO implementations to distinct maintenance and evolution tasks that are recurring in incremental software development.

The changes originated from a variety of sources: the experience of the original developers of Health Watcher including changes they would like to implement (that were actually necessary) and changes from previous empirical studies (Kulesza et al., 2006; Filho et al.; 2006). The remaining changes were created by the students and researchers involved in this study, where certain extensions and improvements that could be applied were identified. This ensured a variety of change sources was used and, as a result, it would not artificially bias the results in favor of one paradigm or another. Before the changes were applied, the original developers of HW were consulted to confirm whether these changes were valid. Each of the change scenarios is summarized in Table 20.

Having implemented the changes, we updated the architecture specification in order to reflect the changes made in the code. Then we applied the concern-diffusion metrics in both non-AO and AO architectures obtained with the changes. The goal was to analyze the impact of the evolution changes in the architecture modularity. The scenario which changes impact most in the architecture was scenario 6 (Table 20). This occurred because this scenario demanded the addition of a number of operations in the interfaces between each connected pair of components. Thus, scenario 6 impacted the boundaries of almost every component in the non-AO and AO architecture. In the non-AO architecture, it affected the ManageDistributedInfo, DistributedInfoServices, ManageInfo and

InfoServices interfaces (Figure 51). In the AO architecture, the change affected the ManageInfo, InfoServices and ServicesDistribution interfaces (Figure 52).

Scenario	Change	Impact
1	Applying the Command design pattern (Gamma, 1995) in order to improve extensibility and ease the process of adding new GUIs.	GUI_Elements
2	Ensure the complaint state cannot be updated once closed to protect complaints from multiple updates.	Business_Rules
3	Encapsulate update operations to improve maintainability using common software engineering practices.	Business_Rules
4	Improve the encapsulation of the distribution concern for better reuse and customization.	GUI_Elements, Distribution_Manager
5	Generalize the persistence mechanism to improve reuse and extensibility.	Business_Rules, Data_Manager
6	New functionality added to support querying of more data types.	GUI_Elements, Distribution_Manager, Business_Rules, Data_Manager

Table 20: Summary of the change scenarios

The measures showed that the persistence concern is more stable in the AO architecture. As the persistence concern is not well modularized in the non-AO architecture (as stated earlier), every operation added in scenario 6 had to address the persistence concern. Each new operation had to consider the persistence-specific exceptional events. The Concern Diffusion over Architectural Operations metric (CDAO) (Chapter 4) highlighted that the number of operations containing the persistence concern in the non-AO architecture increased 38 (from 154 to 192) in the version produced after applying scenario 6. In comparison, the increase that occurred in the AO architecture was just one operation.

7.4. Mobile Media Study

This study involved a software product line (Clements & Northrop, 2002; Pohl et al., 2005), called Mobile Media (Young & Murphy, 2005; Young, 2005). The goal of this study is to evaluate the usefulness of the concern-driven architectural metrics in order to assess the modularity degeneration of the Mobile Media architecture along a series of evolution scenarios. To this end, changes have been undertaken in both AO and non-AO versions of Mobile Media architecture. Architectural metrics have been applied before and after these changes so as to assess the impact of the changes in the architecture modularity.

Mobile Media (Young & Murphy, 2005; Young, 2005) is a software product line (Clements & Northrop, 2002; Pohl et al., 2005) for applications that manipulate photo, music, and video on mobile devices, such as mobile phones. It has about 3 KLOC and was developed based on a previous software product line called Mobile Photo (Young & Murphy, 2005; Young, 2005), conceived at University of British Columbia. In fact, in order to implement Mobile Media, the developers extended the core implementation of Mobile Photo including new mandatory, optional and alternative features.

The features addressed by a software product line are usually described by means of feature models (Pohl et al., 2005). Figure 54 presents a simplified view of the feature model of Mobile Media. In fact, this feature model represents the Mobile Media features after the realization of all change scenarios, i.e. the features addressed by the last release taken into account during this study. The alternative features are just the types of media supported: photo, music, and/or video. Examples of core features are: create/delete media, label media, and view/play media. In addition, some optional features are: transfer photo via SMS, count and sort media, copy media and set favorites. The core features of Mobile Media are applicable to all the mobile devices that are JavaME enabled (Sun Microsystems, 2007). The optional and alternative features are configurable on selected devices depending on the provided API support. Mobile Media was developed for a family of 4 brands of devices, namely Nokia, Motorola, Siemens, and RIM (Young & Murphy, 2005; Young, 2005).

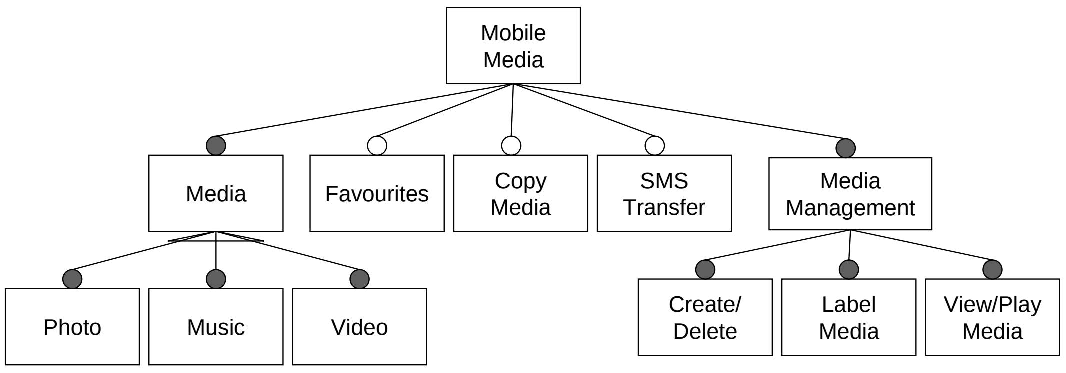

Figure 54: Simplified Mobile Media feature model

The study is divided in four phases: (i) implementation of change scenarios which generated eight successive releases, (ii) definition of the aspect-oriented and conventional architectures of the implemented releases, (iii) mapping of the concerns to architectural elements in the architecture of each release, and (iv) application of concern-driven architectural metrics. In the first phase, an independent group of five post-graduate students was responsible for implementing successive evolution scenarios in both AspectJ and Java versions of the Mobile Media. A total of seven change scenarios were incorporated to the base release of Mobile Media, which led to eight releases, available from (Figueiredo et al., 2008c). Each new release was created by modifying the previous one.

Table 21 summarizes the changes made in order to produce each release. The scenarios comprise different types of changes involving mandatory, optional, and alternative features, as well as non-functional concerns. Table 21 also presents which types of change each release encompassed. Besides, in order to involve typical changes in product line designs, the scenarios were selected based on the consultation with professionals and researchers with long-term experience on the development of software product lines.

Release	Description	Type of Change
R1	Mobile Photo core (Young & Murphy, 2005; Young, 2005)	
R2	Exception handling included (in the AspectJ version, exception handling was implemented according to (Filho et al., 2006, 2007)).	Inclusion of non-functional requirement.
R3	New feature added to count the number of times a photo has been viewed and sorting photos by highest viewing frequency. Label feature changed in order to allow editing the photo’s label.	Inclusion of optional feature. Modification of mandatory feature.
R4	New feature added to allow users to specify and view their favourite photos.	Inclusion of optional feature.
R5	New feature added to allow users to keep multiple copies of photos.	Inclusion of optional feature.
R6	New feature added to send photo to other users by SMS.	Inclusion of optional feature.
R7	New feature added to store, play, and organize music. The management of photo (e.g. create, delete and label) was turned into an alternative feature. All extended functionalities (e.g. sorting, favourites and SMS transfer) were also provided.	Changing one mandatory feature into two alternative features.
R8	New feature added to manage videos.	Inclusion of alternative feature.

Table 21: Summary of the evolution scenarios implemented in Mobile Media

The second, third and forth phases of the study were undertaken according to the procedures described in Section 7.1. The measurement process included different types of concerns, such as optional features, mandatory features, architectural pattern roles and non-functional concerns. Optional features were selected because they are the locus of variation in software product lines and, therefore, they have to be well modularized. The other type of concerns constitute the core of the Mobile Media product line and also need to be investigated in order to assess the impact of changes on them. Table 22 summarizes this study configuration.

Study goals	Analyze the usefulness of the metrics in order to assess the differences of the impact of evolution changes in the modularity of both AO and non-AO architecture versions.
Study Activities	1. Implementation of change scenarios which generated eight releases of both AO and non AO versions of the system; 2. Conception of the architecture description of the implemented releases, according to the procedure described in Section 7.1; 3. Mapping of architectural elements to the concerns, according to the procedure described in Section 7.1; 4. Metrics application (manually); 5. Analysis of the measurement results, assessing the impact of changes in the architecture modularity.
Target architectures	AO and non-AO architectures of the Mobile Media system.
Arch. description approaches	UML 2.0 (OMG, 2005) for non-AO architectures; AO Visual Notation (Section 3.2.2) for AO architectures.
Considered concerns	Sorting media, copying media and setting favorite media (optional features). Labeling media (mandatory feature). Controller role of MVC pattern (Buschmann et al., 1996) (architectural pattern role) Exception Handling (non-functional concern)

Used Metrics	Concern Diffusion over Architectural Components (CDAC);
	Concern Diffusion over Architectural Interfaces (CDAI);
	Concern Diffusion over Architectural Operations (CDAO);
	Component-level Interlacing Between Concerns (CIBC);
	Interface-level Interlacing Between Concerns (IIBC);
	Operation-level Overlapping Between Concerns (OOBC)
	Lack of Concern-based Cohesion (LCC);

Metrics Adaptation	Not necessary

Table 22: Mobile Media study configuration

7.4.1. Mobile Media Architectures

Both non-AO and AO architecture of Mobile Media are mainly determined by the use of the Model-View-Controller (MVC) architectural pattern (Buschmann et al., 1996). Figure 55 presents the non-AO architecture of the eighth release of Mobile Media. Different components realize the three roles of the MVC pattern, namely model, view, and controller. The components whose names end with “Screen” realize the view role. The components whose names end with “Controller” realized the controller role. The other components realize the model role.

Figure 56 presents the AO architecture of the eighth release. In addition to the components realizing MVC pattern, the AO architecture includes a number of aspectual components, namely FavouriteMedia, MediaSorting, ExceptionHandling, SMS, MediaCopy, MediaCapture, PhotoAspect, MusicAspect and VideoAspect. Note that these components are connected to the other components by means of at least one aspectual connector (Section 3.2). The aspectual components do not belong to a specific role of the MVC pattern, since they affects classes in more than one MVC role. The aspectual components were used to modularize optional and alternative features since they are the locus of variability in software product lines. The goal is to enhance the (un)pluggabily of these features by means of aspects. In addition, exception handling was also modularized with aspects, once it is a recurring crosscutting concern.

The eighth release of Mobile Media comprises three alternative features: photo, video, and music. For instance, in the non-AO architecture, the video feature is realized by the components PlayVideoScreen, PlayVideoController, AlbumVideoData and VideoAccessor (Figure 55). Besides these components, the aspectual component VideoAspect contribute to the realization of the video feature in the AO architecture (Figure 56). One of the optional features addressed by the eighth release is SMS transfer. This feature is addressed by the components NetworkScreen and SMSController, and the provided interface ManageSMS of the PhotoViewScreen component in the non-AO solution. In the AO solution, this feature is realized by the same components plus the SMS aspectual component. The ManageSMS interface is not necessary in the AO version.

Other optional features, such as sorting media, copying media and setting favorite media, are not addressed by any specific component in the non-AO architecture. They are addressed by operations and interfaces of the components whose main purpose is to realize other concerns. However, in the AO architecture these features are addressed by specific aspectual components. For instance, the favorite media feature is realized by the FavouriteMedia aspectual component. The copying media feature is realized by the MediaCopy aspectual component and the ControlCopy interface of the components PhotoViewController, PlayVideoController, and PlayMusicController. The graphical descriptions of all eight releases of both non-AO and AO architecture are presented in Appendix A.

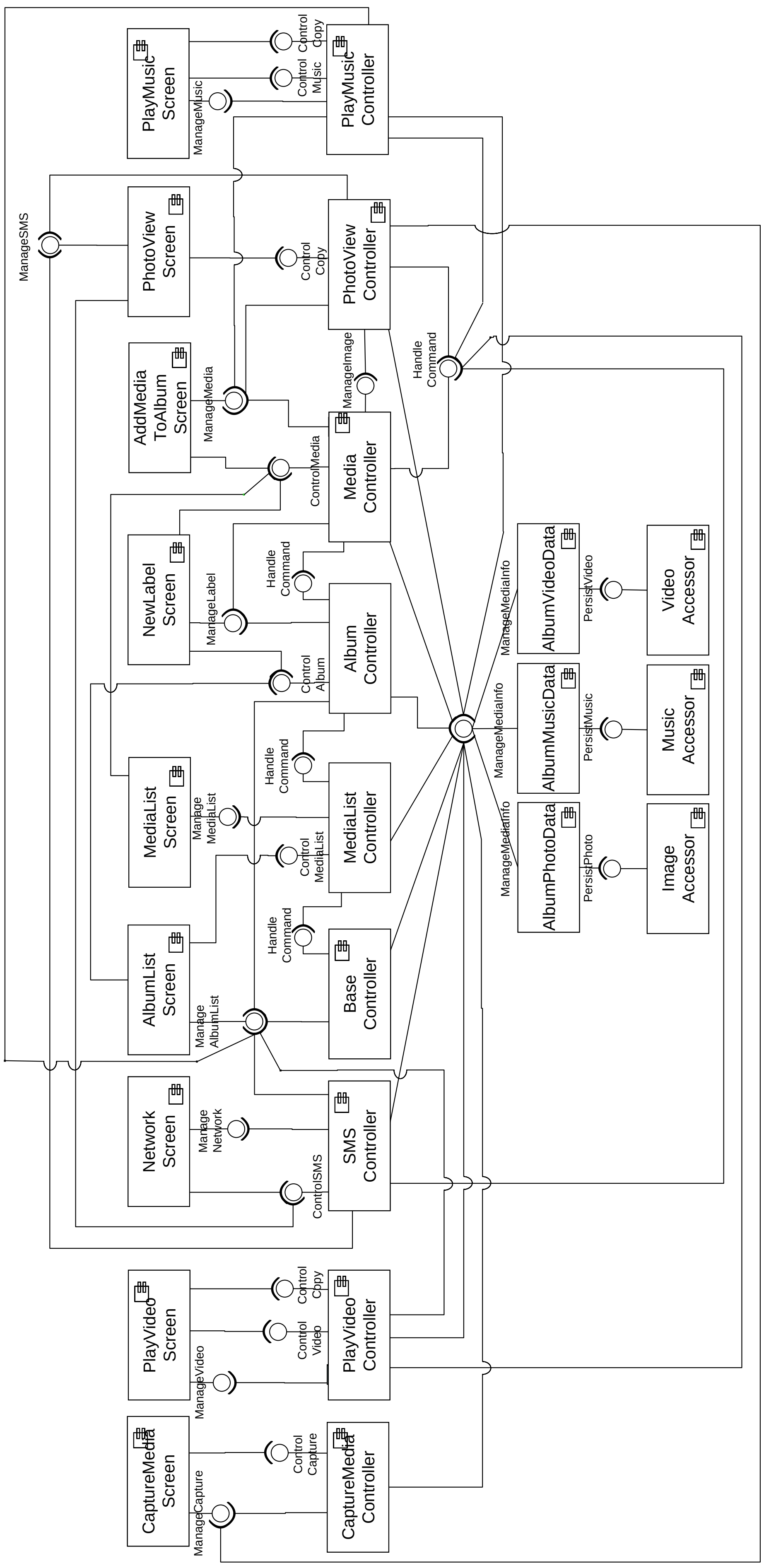

Figure 55: Non-AO architecture of the Mobile Media product line

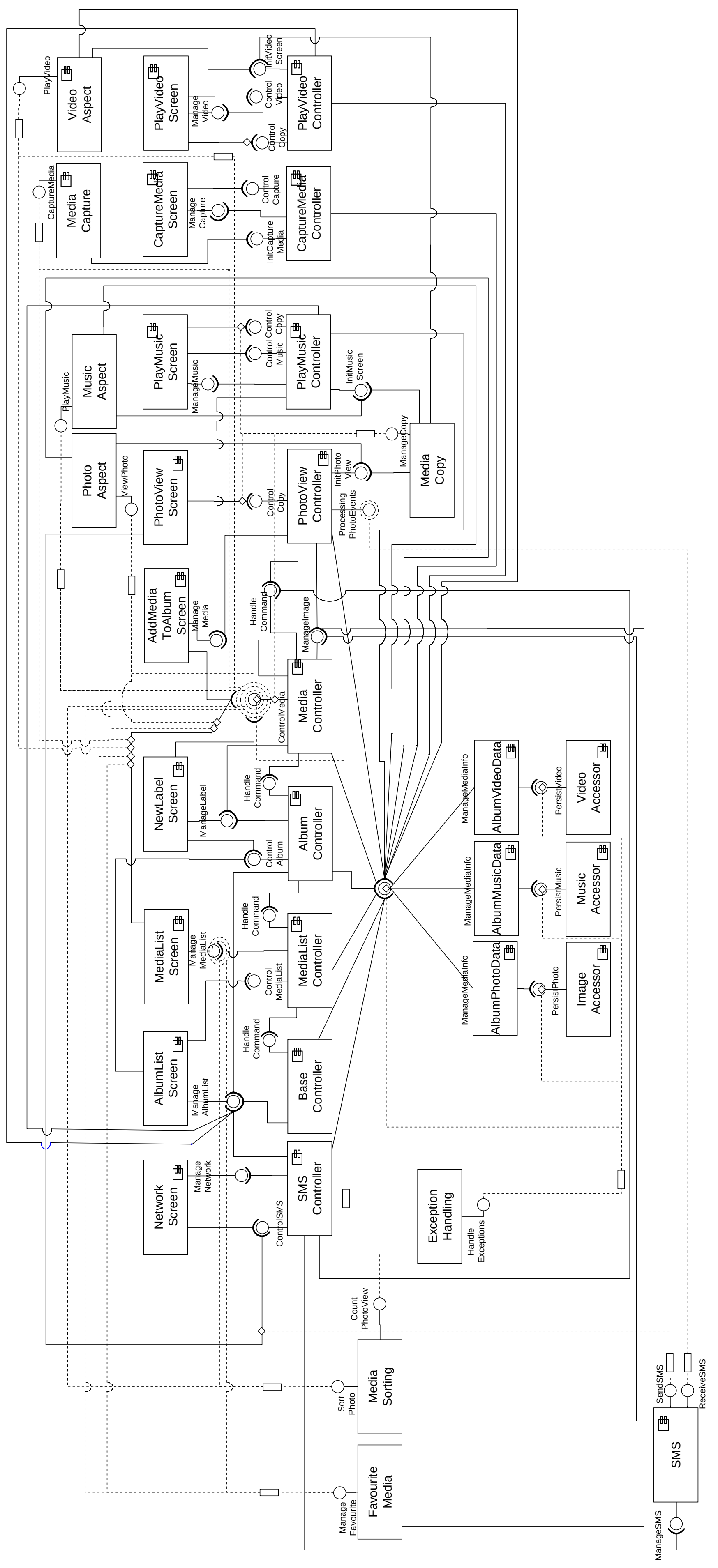

Figure 56: AO architecture of the Mobile Media product line

7.4.2. Results

This section presents the results of the measurement process involving Mobile Media architectures. The non-AO and AO architectures are compared by means of graphics that show the values for each metric along the eight releases. Again, lower values imply better results. From the analysis of concern diffusion (Section 4.3.1) and interaction between concerns (Section 4.3.2) measures, three groups of features naturally emerged with respect to which type of modularization paradigm performed better.

AO architecture succeeded for optional features

This group encompasses the analyzed optional features: sorting media (hereafter referred as sorting), copying media (hereafter referred as copy), and setting favorite media (hereafter referred as favorite). A common characteristic of these features is that the modularity of their design is stable in both AO and non-AO architectures, in the sense that it did not degenerated because of the undertaken changes. Figure 57 shows the results of concern diffusion and interaction between concerns metrics for the favorite feature as a representative of this group. We can observe from this figure that the number of architectural elements affected by the favorite feature does not vary along the eight releases in both AO and non-AO architectures. In addition, the number of concerns with which the favorite feature interacts is not expressively impacted by the changes.

Although both AO and non-AO solutions presented similar degree of modularity stability, the AO solution performed better in terms of scattering and tangling (Figure 57). The favorite feature is spread over fewer components and interfaces (CDAC and CDAI) and tangled with fewer concerns in the AO architecture (CIBC and IIBC). This occurred because the AO mechanisms effectively transferred all the elements in charge of realizing this feature from conventional components (MediaListScreen, MediaController and PhotoViewController) (Figure 55) to only one aspectual component (FavouriteMedia) (Figure 56). This almost totally separated this feature from the other concerns. The only concern still interacting with this feature is the exception handling concern. The reason for that is because the aspectization of exception handling was not able to modularize all the elements related to this concern and, as a consequence, exceptional events are still handled by some operations in the FavouriteMedia component.

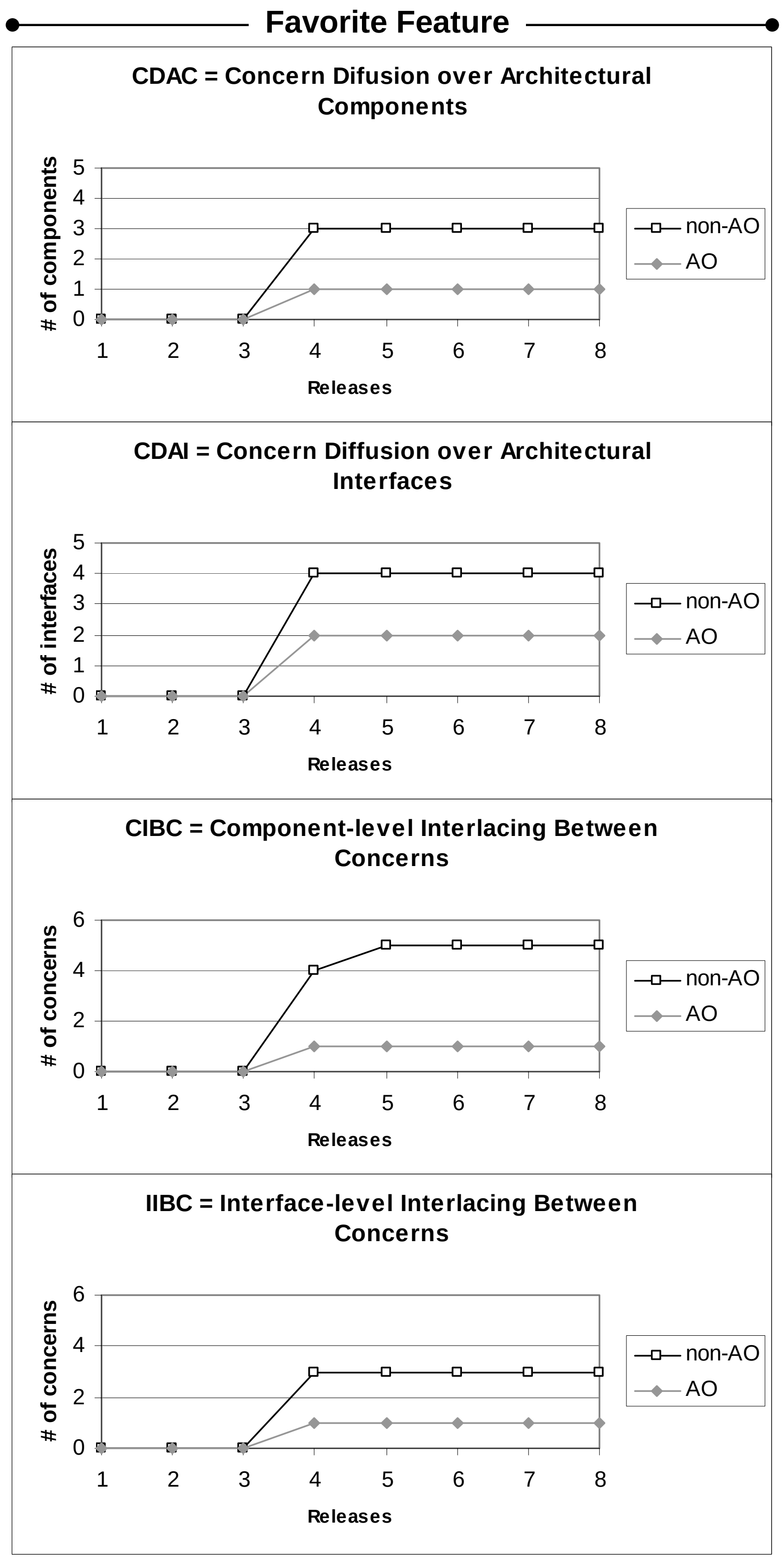

Figure 57: Concern Diffusion and Interaction between Concerns metrics for favorite feature

AO solution was harmful to modularity of Exception Handling

Exception handling tended to present slightly superior modularity stability in the non-AO architecture. Figure 58 shows metrics results for the exception handling concern. We observe that the modularity of this concern is more stable in the non-AO version, since this feature is spread over fewer components (CDAC) in this solution. Besides, the difference increases throughout the releases due to the rising of CDAC in the AO solution. The CDAI and CDAO results show the same trend.

Although, exception handling was aspectized by means of the ExceptionHandling aspectual component (Figure 56), this aspectization did not completely localize and separate the exception handling from the other concerns. The aspect-oriented solution was not able to eliminate the interaction among exception handling and the other concerns, including optional and alternative features. This can be observed by the last graphic in Figure 58, which shows the results for the Component-level Interaction between Concerns metric. Note that the degree of interaction between the exception handling and other concerns is the same in both AO and non-AO architectures. Therefore, as new optional and alternative features were included over the different releases, the number of components that contains exception handling increased.

The reason for this difference in favor of the non-AO solution is that the number of architecture elements included over the releases is higher in the AO version. Adding optional or alternative features (releases 4 to 8) required the introduction of more components, interfaces and operations in the AO version because new aspectual components have to be included in addition to the conventional components realizing the features. For instance, the introduction of the optional SMS feature (release 6) in the non-AO required the inclusion of the NetworkScreen and SMSController components (Figure 55). Both components encompass elements related to exception handling. In the AO version, in addition to these same two components, it was also necessary to add the SMS aspectual component. The SMS aspectual component also encompasses exception handling elements. As a consequence, the number of components encompassing the exception handling concern increased more in the AO version than in the non-AO one. As a conclusion, the results of this group indicate that using aspects to modularize exception handling, optional and alternative features in the investigated product line negatively impacted on the modularity of exception handling.

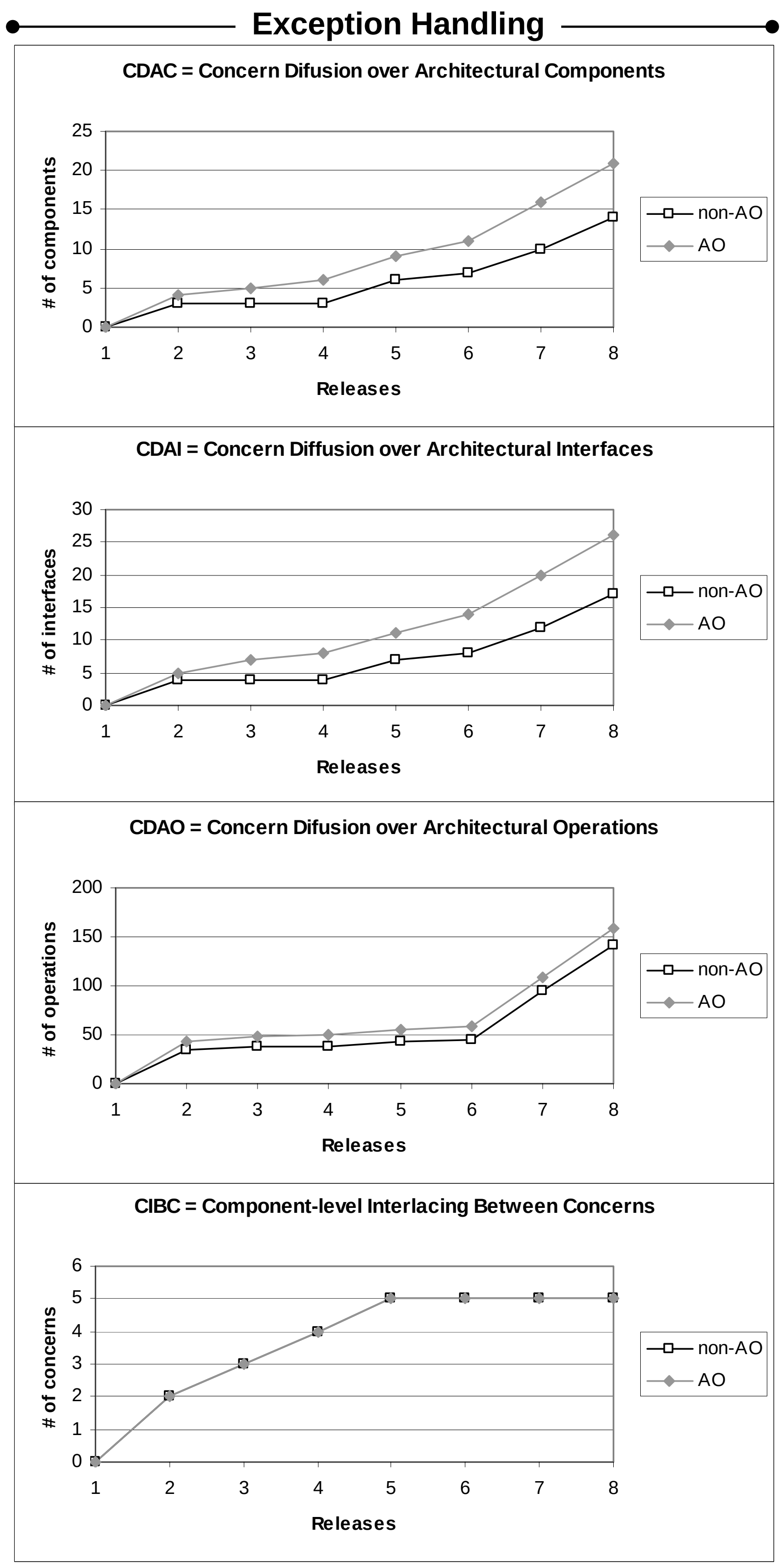

Figure 58: Concern Diffusion and Component-level Interlacing between Concerns metrics for the exception handling concern

AO improved tangling of mandatory features

Mandatory features, which form the core of software product lines, were not aspectized in this study. As mentioned before, aspect-oriented mechanism had been used in the Mobile Media software product line in order to modularize optional and alternative features. As a consequence, the degree of scattering of each of the assessed mandatory features evolved in a similar way in both AO and non-AO architectures. This can be observed from graphics shown in Figure 59.

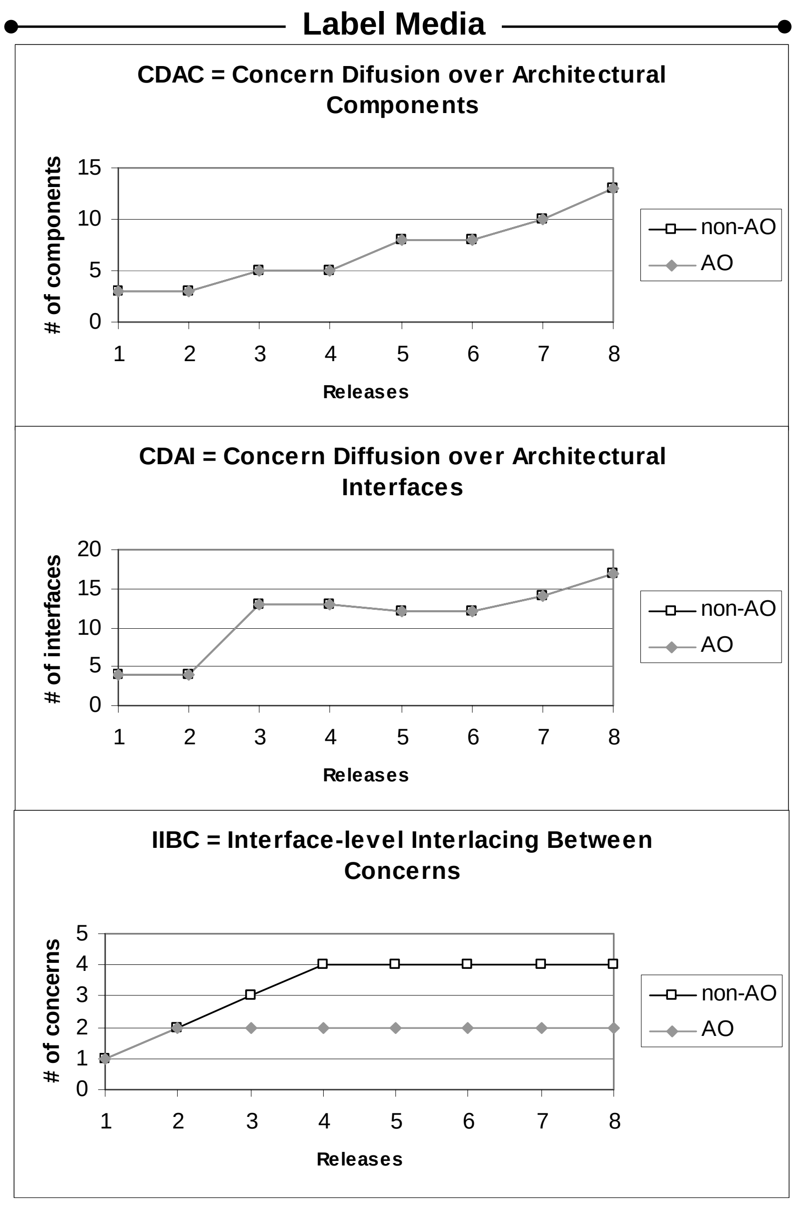

Figure 59: Concern Diffusion and Interface-level Interlacing between Concerns metrics for the label media mandatory feature

The first two graphics present the results of concern diffusion metrics for the label media mandatory feature. These graphics show that the number of components and interfaces realizing the label media feature is the same in both AO and non-AO architecture. Besides, this number evolves in the same way throughout the eight releases. However, the last graphic in Figure 59 shows that the AO solution was able to decrease the tangling of the label media feature with other concerns. This graphics shows that this feature is interlaced with fewer concerns in the AO architecture. This is shown by means of the Interface-level Interlacing between Concerns metric, but the results for component-level interlace are similar. Another concern which results show the same trend of label media is the controller role, which also forms the core of the Mobile Media product line.

This section presented the third study for evaluating the usefulness of the concern-driven architectural metrics. Together with the studies presented in previous section, the evaluation involved four systems: MobiGrid, AspectT, Health Watcher and Mobile Media. These studies showed evidences that concern-driven architectural metrics represent a promising mechanism for complementing conventional metrics and improving modularity quantitative assessment.

7.4.3. Discussion

In the previous section, we presented the results of the concern-driven measurement process for the Mobile Media architecture. Here, we discuss some specific issues and lessons learned during this study. Some of these issues are also related to results obtained in the other presented studies.

Complementary metrics

The study with Mobile Media showed that concern diffusion and interaction between concerns metrics complement each other in order to identify the crosscutting nature of concerns. The concern diffusion metrics assess the degree of scattering of a given concern, while the metrics for interaction between concerns measure the degree of tangling of a concern. For instance, the Concern Diffusion over Architectural Components (CDAC) and Concern Diffusion over Architectural Interfaces (CDAI) metrics showed that the label media feature has the same degree of scattering in both AO and non-AO Mobile Media architecture (Figure 59). The Interface-level Interlacing between Concerns metric complement this information showing that, in spite of having the same degree of scattering in both architectures, label media feature is tangled with more concerns in the non-AO architecture. This confirmed similar results obtained in the study involving the Health Watcher system (Section 7.3). The information about tangling can also be obtained by the Lack of Concern-based Cohesion (LCC) metric (Section 4.3.3). However, since this metric provides the results per component, each component has to be checked in order to compute the tangling of a concern.

Concern Diffusion over Interfaces and Operations

Based on the results gathered in the two first studies, which involved the MobiGrid, AspectT and Health Watcher systems, it seemed that it was enough to use either the Concern Diffusion over Architectural Interfaces metric or the Concern Diffusion over Architectural Operations metric. This is because in those studies the results of these metrics showed that every time a concern was spread over a high number of interfaces, it was also spread over a high number of operations, and vice-versa. Table 16 (Section 7.3.2) showed, for instance that four concerns (distribution, concurrency, persistence and exception handling) are scattered over more interfaces in the non-AO architecture than in the AO solution. All these concerns are spread over more operations in the non-AO architecture as well. The other two concerns GUI and business are scattered over the same (or nearly the same) number of interfaces in both AO and non-AO architectures. The amount of operations where these concerns are spread over is the same in both solutions as well (Table 16 - Section 7.3.2).

Nevertheless, the Mobile Media study contradicted this observation. This study showed that there are cases in which, although spread over few interfaces, a concern can be spread over a high number of operations. This can be observed from the results for the favorite feature. The second graphic in Figure 57 shows that the number of interfaces where the favorite feature is scattered over is twice higher in the non-AO architecture than in the AO one. However, this same feature is spread over slightly more operations in the AO architecture, as shown in Figure 60. This occurred because the aspectual component which encapsulates the favorite feature comprises additional operations to capture the context of the other components affected by it.

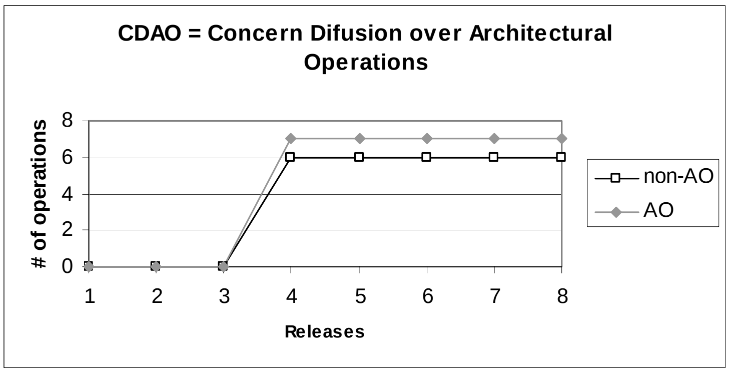

Figure 60: Concern Diffusion over Architectural Operations metric for the favorite feature

7.5. Study Constraints

This section discusses some constraints related to the studies presented in this chapter. First, we should also emphasize that the conclusions obtained from our studies are restricted to the specific assessed systems, chosen architecture alternatives and analyzed concerns. In other words, results regarding advantages and drawbacks in modularizing certain concerns with aspect-oriented abstractions may not be directly generalized to other contexts. However, these studies allowed us to make useful assessments of whether the use of aspects for the modularization of architectural concerns would be worth studying further. In addition, the studies also allowed us to make useful evaluation about the applicability and usefulness of the architectural metrics.

Another issue that limits the conclusions is the fact that the process of assigning concerns to design elements, required for the concern-driven metrics computation, directly impacts on the measurement results. Variations on this process could lead to variations on the measurement outcomes. As mentioned in Section 7.1, in order to make this process more systematic, we followed a guideline which states that assign a concern to a design element if the complete removal of the concern requires with certainty the removal or modification of the element. Also, we have consistently observed in our studies that, as expected, mapping concerns to architecture elements is easier and less time-consuming than mapping to elements of detailed design and source code due to the higher level of abstraction in architecture descriptions.

In addition, we took some measures to support the mapping of concern to design elements, such as: (i) “pair mapping”, where the assigning of concerns to design elements was done by two people assisting each other, and (ii) consultation of the actual system developers, when possible. These procedures are in line with the procedures followed by other researchers working on concern-based analysis (e.g. Eaddy et al (2007)). Assessing the impact of these issues on the reliability of the concern assignment is out of the scope of this thesis. It would demand more controlled experiments with different nature of the ones we undertook.