First reviewer's review:

Summary of the submission <<<

This paper proposes a novel mechanism, called instance pointcuts, to create and

maintain sets of objects based on events in their life-cycle. Instance

pointcuts can be flexibly refined and composed, facilitating reuse and

evolution. Based on an example, the authors analyze the weaknesses of

object-oriented and aspect-oriented languages with regards to managing objects

based on their life-cycle. Based on this analysis, the authors develop their

proposal of instance pointcuts by means of the AspectJ language. Then, they

report on the implementation of their mechanism based on EMFText and ALIA4J.

Finally, the authors validate their approach by means of a discussion.

Evaluation <<<

The paper is quite well written and presents here and there interesting

insights. The main problem I have --maybe as an informed outsider-- is that I

don't see what the practical relevance or the theoretical insight of this work

is. Although I see that the instance-pointcut solution is smaller and more

declarative than the AspectJ solution, I had a hard time to imagine meaningful

use cases of such a mechanism. The point is that this approach adds even more

syntax to an aspect-oriented language, and this has to be justified by as many

beneficial use cases as possible. The very last paragraph of the paper mentions

some promising directions: design patterns. If the authors could show that

instance pointcuts could simplify multiple design patterns, I would be more

convinced. A related issue is certainly the almost missing evaluation. I would

like to see of often instance pointcuts are applicable in practical systems and

what we gain (a case study is certainly a good first step).

Detailed comments:

Section 1:

- The introduction fails to motivate the need for instance pointcuts. The idea

of switching roles at runtime (and supporting this by the langauge) is

interesting, but they authors do not follow up on this idea.

- It is hard to grasp what is meant by the fact that an instance pointcut

\*maintains\* a set of objects, especially, for readers with AspectJ background

(not to speak of readers without AspectJ background).

***We have improved the wording and added an explicit definition of what we mean by maintaining an instance pointcut’s object set.***

- The difference between refinement and composition via set operations is not

clear in the introduction.

***We added an explanation.***

Section 2:

- It seems that Listing 1 (and other listings, e.g., Listing 2) contain errors,

for example, in Line 23.

***We fixed these errors***

- The problem definition at the end of Section 2 is quite fuzzy. Same for the

overall motivation.

***We turned the problem statement to a separate section and formulated it more to the point. Section 2 now fully focuses on discussing the motivating example.***

Section 3:

- This section contains a lot of details about the mechanism and rules of

instance pointcuts. However, sometimes the design decisions and rules are not

well motivated. For example, "A sub-expression contains at least one expression

element and at most two". Why?

***We added explanations for several design decisions.***

- Section 3.3, describes two usage scenarios of instance pointcuts. The

relation to the initial motivation of introducing instance pointcuts is

unclear. Is it a good idea to call an aspect from the base code? This deserves

a discussion.

***We did not write that the instance pointcuts should be used from the base code. Nevertheless, in order to avoid confusion, we added an explanation on the visibility of instance pointcuts at the end of the introduction to Section 3.3.***

Section 4:

- The connection between EMFText and ALIA4J as bases for implementation is

unclear. Is there a tool that can be used like the AspectJ compiler?

***We improved the explanation of the interplay between EMFText and ALIA4J.***

- This section contains too much generated code. I would prefer a more general

and abstract way to describe the (rewriting) semantics of instance pointcuts.

***We believe that the description in Section 4 (was Section 3 in the previous version) acts as such an abstract description. The focus of our compilation section should be a truly operational definition. For this reason we did not change its content.***

- In Section 4.3, the authors talk about interference between instance

pointcuts and plain AspectJ constructs. Especially, ordering advice execution

was always a problem with AspectJ (Roberto E. Lopez-Herrejon, Don S. Batory,

Christian Lengauer: A disciplined approach to aspect composition. PEPM 2006:

68-77). It seems that the situation is now even worse. Please discuss.

Section 5

- As said previously, the validation is rather weak. There is no support for

statements like "Such reuse and composition mechanisms are beneficial for

software evolution..."

***We have toned down this statement and moved it to the conclusion section.***

- Looking at the checking rules, I wonder whether the underlying type system is

sound and complete.

***This is a good point and we will pay attention to it when we get to implement the proposed checks. For now, we have re-worded the introduction of the checks to make clear that they are work in progress.***

\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*

Second reviewer's review:

Summary of the submission <<<

The paper describes a new type of point cut expression that is used to

determine objects of a class that are in particular phases. The expressions are

referred to as instance point cuts.

Evaluation <<<

The idea seems interesting, but it is very weakly motivated. In particular it

was not clear to me how theses expressions and ability to reuse and compose

these specifications significantly improves the quality of the implementation

as claimed. The validation section provides some qualitative arguments, but

they are weak. Still the ideas are interesting enough to be presented in a

conference setting.

The claims about improved quality made in the introduction need to be scaled

back, especially given that no formal validation is provided in the paper.

***We have done so.***

\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*

Third reviewer's review:

Summary of the submission <<<

The paper sets out to define instance pointcuts, the abstract says:

Selecting objects according to such phase shifts results

in scattered and tangled code. In this study we propose

a new kind of pointcut, called instance pointcuts, for creating

and maintaining sets of objects according to events in

their life-cycle; these events are selected with pointcut-like

specifi cations.

And further proposes to do it in a way that is compatible with AspectJ.

I found the paper very confusing to read the first time, because I kept

expecting them to turn the corner and make these things into pointcuts in the

AspectJ sense of the term. But while these things may be useful, these aren't

AspectJ pointcuts at all.

Rather they are a whole new join point mechanism, that partly builds on

AspectJ's poincuts and advice mechanism. The join points appear to be objects

or sets of objects, as well as events at which the objects join/leave the set.

So what a programmer can't do easily is take a instance pointcut like onSale

and use it together with other AspectJ pointcuts in a combined pointcut. It can

be made to work using the if() pointcut of course.

So at the very least the paper needs some work on terminology. Technically it

would be enough to say "when this paper says pointcut it means the the more

general concept of some means of identifying join points". Its pretty important

to get this right. But it might be better to give these things a more

descriptive name, like state crosscuts or something.

***We have added a respective explanation in the end of the introduction.***

But the problem may be worse than just terminology. If these things can't be

composed easily with AspectJ pointcuts then they have to be useful on their

own. Its not clear to this reviewer from the examples in the paper that this

mechanism is useful enough to warrant language inclusion as opposed to just

coding it by hand, perhaps with a library in AspectJ or some other AOP

language.

Having said that, I wonder whether there isn't some missing work in type

states?

\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*

Fourth reviewer's review:

Summary of the submission <<<

The paper presents a new language feature "instance pointcuts" as an

extension to the AspectJ language. Instance pointcuts enable

programmers to capture instances of objects based on events of

interest during their life-cycle at runtime. A new pointcut descriptor

is presented along with an example based on an online retail system.

Evaluation <<<

The paper attempts to make a case for a new pointcut designator and

presents a use case to support the need for this language extension.

I was not convinced that the use case is general enough

to require a language extension. The first issue is the design for

the online store. It is typically the case that there is a

relationship between product and vendor. It is however typical that

the relationship is bi-directional. Furthermore, the class

ProductManager seems to contradict its name. ProductManager is in

fact a VendorManager since it manages vendors.

My expectation was to have a product manager role that manages

products not items. Also discounts can be modelled as parts of

Products (not vendors). But in a larger system, adding discounts

implies a special relationship between product and price modelled

differently, typically in a DB, that can be mapped to an OO layer

(e.g., hybernate or carbonado) but still the logic on which discount

applies when and by how much is applied at the object's creation

time. The decisions on what discount is applied when and where is

encapsulated in a rules engine that can get complex. It is not clear

(to me) how an instance pointcut will assist in such scenarios.

Looking at the proposed problem (and not the use case) the paper

does specify that instance pointcuts might have more uses in the

context of design patterns. This seems to be left as future work or

something that is, at least in this version of the paper, not

discussed at all. Even at a high level. A larger example set

analyzing possible solutions using instance pointcuts along with the

benefits and shortcomings would substantially improve the paper.

***The provided examples should act as illustrations of our approach. We do not claim that our approach is applicable to every implementation of an online store.***

Definition 1 states:

 h(v) = min (f(x), g(x))

x is free here. Was it supposed to read

 h(v) = min (f(v), g(v))?

I have the same comment for

 i(z) = max(f(x), g(x))

x is free again. Was it supposed to read

 i(z) = max(f(z), g(z))

***The reviewer is correct. We have fixed this mistake in the paper.***

Listing 16, lines 7 - 9 are these lines supposed to be comments?

There is no comment marker in the listing itself, the font seems to

indicate that these lines are comments.

***We have turned this text into comment.***

\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*=--=\*

Fifth reviewer's review:

Summary of the submission <<<

This paper presents an approach called instance pointcuts which is used to

group objects of a specific type that belong to a specific phase. A phase is

determined by start events and end events, i.e., operations performed on the

object. The grouping of the objects allows phase-specific behavioral extensions

to be added to a system. Instance pointcuts can be reused and composed. An

implementation of instance pointcuts in ALIA4J and a number of examples are

discussed as validation of the approach.

Evaluation <<<

Comments:

This is a technically strong paper that presents complex subject matter quite

clearly. Its claims need to be reduced a little bit to avoid too much

conjecture (e.g., "These features make instance pointcuts easy to evolve

according to new requirements." and "Such reuse and composition mechanisms are

beneficial for software evolution since they make it easy to create tailored

variations according to new requirements."). No evidence supporting these

statements is given in the paper. The paper shows that it is possible to

specify and compile instance pointcuts, but how effective, useable, and

evolvable they are is not investigated through any kind of empirical

validation. The qualitative validation of those factors is also rather weak and

would have to be strengthened significantly, if statements about evolvability

etc want to be made.

The related work section does a reasonable job in comparing similar work at the

AOP level. However, it is not clear whether an AOM approach would not be more

effective and easier to understand. In essence, instance pointcuts allow a

specific set to be defined to which instances are added and removed at certain

points in time (i.e., when methods of the object are called). In standard AOM

techniques such as Kompose or RAM, it is straightforward to define an aspect

that encapsulates a set including add/remove operations (and many other ones

such as, e.g., checking for empty sets). It is also straightforward to bind the

add/remove events to specific methods, hence adding/removing instances at the

appropriate times. Consequently, boilerplate code only needs to be specified

once and is essentially hidden from the user of the "set" AOM aspect. This is

similar to what the paper claims instance pointcuts can achieve, but at a

different abstraction level. Please discuss this in the paper. Another approach

could be to annotate the protocol state machines of the object with add/remove

events and the corresponding set. Again, this would result in a similar effect

as instance pointcuts (but arguably easier to understand) and should be

discussed.

Minor:

Please show Discount and SurpriseDiscount in Figure 1 because they are used in Listing 1.

***done***

p3 - "Checks like determining if the set is empty cannot be performed.":

surpriseDiscount.isEmpty() performs such a check and could be provided by the

SDiscount aspect.

***We meant to write that emptiness cannot be proven or disproven statically. We improved the writing.***

p4: Briefly summarize the shortcomings discussed in [11].

***done***

p6 - "h(v) = min(f(x); g(x))": replace x with v (twice)

p6 - "i(z) = max(f(x); g(x))": replace x with z (twice)

***done***

p8 - static instance pointcut in "Type Refinement" section: should not "Type2"

be "Type" as the type specification has to be the same for "ipc" and "ipc1" (at

least according to the BeautyProduct example in section 3.2.1).

***The reviewer is right. We fixed this error.***