Diagnosis of the Significance of Inconsistencies in Object-Oriented Designs: A Framework and Its Experimental Evaluation

(Draft)

George Spanoudakis, Hyoseob Kim
Department of Computing, City University
Northampton Square, London, EC1V 0HB
E-mail: (gespan|hkim)@soi.city.ac.uk

ABSTRACT: This paper presents: (a) a framework for assessing the significance of inconsistencies which arise in object-oriented design models that describe software systems from multiple perspectives, and (b) the findings of a series of experiments conducted to evaluate it. The framework allows the definition of significance criteria and measures the significance of inconsistencies as beliefs for the satisfiability of these criteria. The experiments conducted to evaluate it indicate that criteria definable in the framework have the power to create elaborate rankings of inconsistencies in models.

KEYWORDS: diagnosis of inconsistencies, UML, OCL, Dempster-Shafer beliefs

1. Introduction

The need to describe complex software systems from different design perspectives, such as those of the static structure and the interactions of the components of a system, may result in the construction of many partial system design models (or simply "models" henceforth). These models may be constructed independently by different designers, may advocate specific modelling angles and may reflect disparate perceptions of these designers. As a result, they may be inconsistent with each other.

Inconsistencies occur when partial models refer to common aspects of the system under development and make assertions which violate consistency rules applicable to these aspects (Hunter and Nuseibeh, 1998; Spanoudakis and Finkelstein, 1996; Spanoudakis and Zisman 2001).

As an example consider an object-oriented design model that consists of an object interaction diagram and a class diagram. Assume also a consistency rule requiring that for any message received by an object in the interaction diagram, an operation with the same signature as the message must have been defined for one of the classes of the object in the class diagram. In this model an inconsistency would arise if there was a message with no counterpart operation, thus violating the above consistency rule.

Inconsistencies are inevitable in software development (Schwanke and Kaiser, 1998). And, although they will have to be settled eventually, they may need to be tolerated temporarily to give designers a chance to work independently developing their own parts of a model without the need for continual reconciliation (Hunter and Nuseibeh, 1998; Spanoudakis and Finkelstein, 1996). In settings providing freedom for groupwork, it is important to be able to *diagnose* the significance of an inconsistency in order to decide when and with what degree of priority it has to be settled. In one of the experiments reported in Section 5.1, we detected 90 violations of the consistency rule mentioned above. In such cases having a mechanism to assess the significance of inconsistencies and order them by this significance would be undoubtedly useful.

In this paper, we describe a framework that we have developed to support the assessment of the significance of inconsistencies in object-oriented software design model expressed in UML (OMG, 1999) and present the main findings of a set of experiments that we conducted to evaluate it. A description of the framework at an earlier stage of its development is given in (Spanoudakis and Kasis, 2000).

The main premise of the framework is that the significance of an inconsistency depends on the significance of the model elements that give rise to it for the model. The framework defines a set of *characteristics* which indicate the significance of the main kinds of elements in UML models. The assessment of whether or not an element has a particular characteristic in a model is approximate; the framework incorporates *belief functions* measuring the extent to which it may be believed from its modelling that an element has the characteristic. The need for approximate reasoning arises because it cannot be guaranteed that the model provides a consistent, complete and accurate description of the system it describes at the different stages of its evolution. In addition, it cannot be guaranteed that the element will retain the characteristic in the next version of the model.

The framework presented in this paper has been developed as part of a semi-automated method that we have developed to assist software developers in managing inconsistencies in object-oriented software design models, called "reconciliation". Reconciliation supports the entire range of the activities of what has been termed in the literature as "inconsistency management", including the detection of overlaps and inconsistencies in software models, the diagnosis of the significance of inconsistencies, and the handling of detected inconsistencies (Finkelstein et al., 1996; Spanoudakis and Zisman, 2001). A full description of this method is however beyond the scope of this paper and can be found in (Spanoudakis and Finkelstein, 1997; Spanoudakis and Kim, 2001).

The rest of this paper is structured as follows. In Section 2, we introduce the characteristics which indicate the significance of model elements and the belief functions associated with them. In Section 3, we establish a scheme for expressing consistency rules and significance criteria which determine the characteristics that the elements which violate these rules must have for the violations to be significant. In Section 4, we give an example of how to use these criteria to evaluate the significance of inconsistencies and rank them. In Section 5, we present the results of an experimental evaluation of the framework. In Section 6, we overview related work in Section 7 we summarize the framework and present directions for further work on it. The paper has also an appendix which overviews the statistics used in the rank correlations discussed in Section 5.2.

2. Characteristics of significant model elements

The UML models assumed by our framework can be composed of any number of *class* and *sequence diagrams*. Class diagrams specify the static structure of, and the relationships between the classes of a system. Classes can have *attributes*, *operations*, and be related by *associations* and *generalisation* (*Is-a*) relations. Sequence diagrams specify *interactions* between the instances of these classes (the terms "sequence diagram" and "interaction" are used synonymously in the rest of the paper). An interaction consists of a set of messages exchanged between objects to deliver part of the functionality of a system. A complete description of the semantics of these kinds of UML model elements is beyond the scope of this paper and may be found in (OMG, 1999).

In our framework, the significance of the above kinds of UML model elements is indicated by six characteristics: the *genericity* and *coordination capacity* of classes, the *functional essentiality* of attributes and association ends, the *charactericity* of operations, and the *functional dominance* and *coordinating capacity* of messages. These characteristics are described below.

2.1 Class genericity

In software models, classes with numerous subclasses normally specify interfaces (i.e. sets of operation signatures) for groups of services which are provided by their subclasses and the internal state of the instances of these subclasses which is required to realise the services. In effect, such generic classes provide a basis for specifying clients capable of using the services without knowing the exact class which provides them. An inconsistency involving the specification of a generic class is significant since it may affect both its subclasses and the clients that use its services.

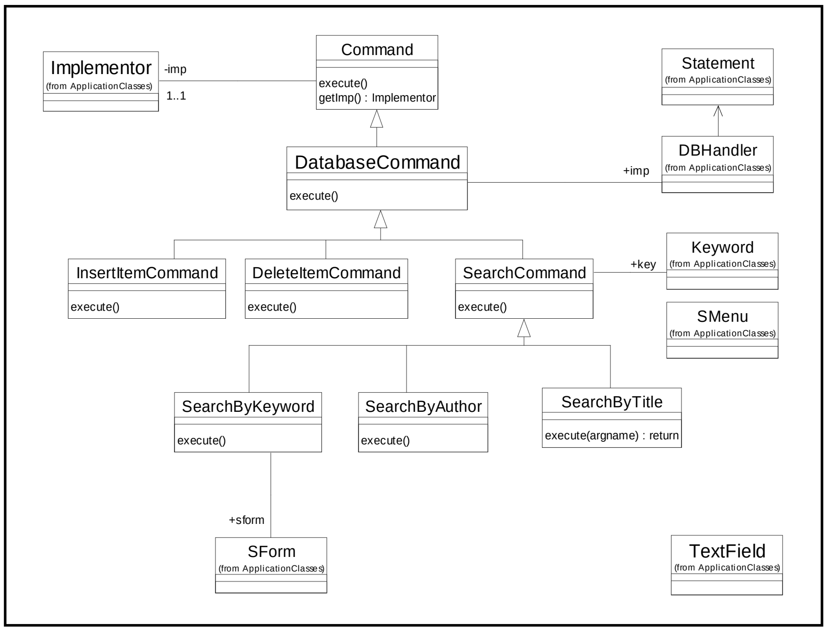

Figure 1. *UML class diagram for a library system*

The belief to the genericity of a class in our framework is measured as the likelihood of an arbitrary class in a model being a subclass of it:

Definition 1: The belief to the genericity of a class *c* in a model M (denoted by the predicate *gen-c(c)*) is defined as:

m_1(gen-c(c)) = |c.Sub*| / | AllClasses(M) – {c}|[1]

m_1(¬gen-c(c)) = 1 – m_1(gen-c(c))

where

- *AllClasses(M)* is the set of all the classes in model M
- *c.Sub** is transitive closure of the subclasses of *c*

Figure 1 shows a generalisation hierarchy of command classes for a library system (see Section 2.2 below) which has been modeled following the command pattern in (Gamma et al., 1995). The degrees of belief in the genericity of the command classes *Command*, *SearchCommand*, and *SearchByKeyoword* in this hierarchy generated by m_1 (assuming that the classes in Figure 1 are the only classes of the model involved) are: m_1(gen-c(Command)) = 0.5, m_1(gen-c(SearchCommand)) = 0.21, m_1(gen-c(SearchByKeyword)) = 0

2.2 Coordination capacity of classes

Some classes in the design of a system may have a coordination capacity, that is they may exist to coordinate interactions between other classes. Coordinating classes are very important in a design since they encapsulate protocols of interactions between the classes they coordinate and, thus, they appear in numerous design patterns (e.g. *mediator*, *observer*, *facade* (Gamma et al., 1995)). An inconsistency involving a coordinating class is important since it is likely to affect all the classes and the interactions which are coordinated by this class.

A common characteristic of coordinating classes across all the different coordination patterns that they may realise is that they send messages to or receive messages from all the classes that they coordinate. Drawing upon this observation, we measure the belief to the coordination capacity of a class *c* in a set of interactions *S* as the likelihood that an arbitrary class in *S* will be communicating with *c*:

Definition 2: The belief to the coordination capacity of a class *c* in a subset *S* of the interactions of a model (denoted by the predicate *coord-c(c,S)*) is defined as:

m_2(coord-c(c,S)) = |Com(c,S)|/|Classes(S)–{c}|

m_2(¬coord-c(c,S)) = 1 – m_2(coord-c(c, S))

where

[1] The expression |S| denotes the cardinality of the set S.

- *Com(c,S)* is the set of the classes whose instances send messages to or receive messages from the instances of *c* in the interactions of the set *S* excluding *c*
- *Classes(S)* is the set of the classes which appear as receivers or senders of messages in the interactions of *S*.

As an example of using m_2 to measure the coordinating capacity of classes consider the sequence diagrams of Figures 2 and 3.

The sequence diagram of Figure 2 shows an interaction between the classes of the library system whose static class structure was specified in Figure 1. This interaction takes place to allow the user to select one of the search options available from the system. More specifically, the system offers the options of searching by keywords in the title, author or the keywords associated with library items. A search menu (*SMenu*) is used to activate the various search options offered by the system. These options are modeled (and operationalised) by the command classes *SearchByKeyword*, *SearchByAuthor*, and *SearchByTitle* (in the sense of command classes in (Gamma et al., 1995)).

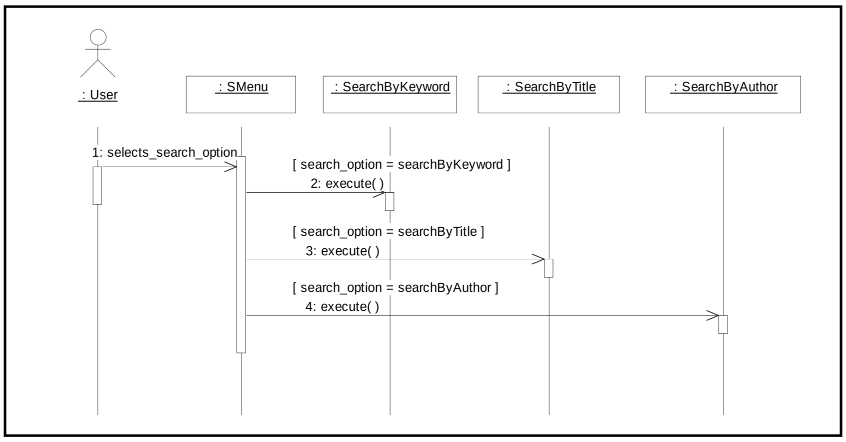

Figure 2. *I_1 - Interaction for selecting a search option*

The sequence diagram of Figure 3 shows the interaction that takes place when the system is used to search for library items by keywords. As shown in the diagram, when the command class *SearchByKeyword*. is activated to execute the operation *execute()*, it displays a search form (see message *setVisible(True)*), set itself as a listener of events related to a text field of this form (see message *addActionListener(sbk)*), gets the contents of the text field (see message *getText()*) when it is notified that the user has typed something in it (see message *actionPerformed(event)*), constructs a string representing an SQL query (see message *formQuery()*), and invokes an operation in the class *DBHandler* (i.e., a database driver) to execute this query *(*see message *executeQuery(String,OCol)*).

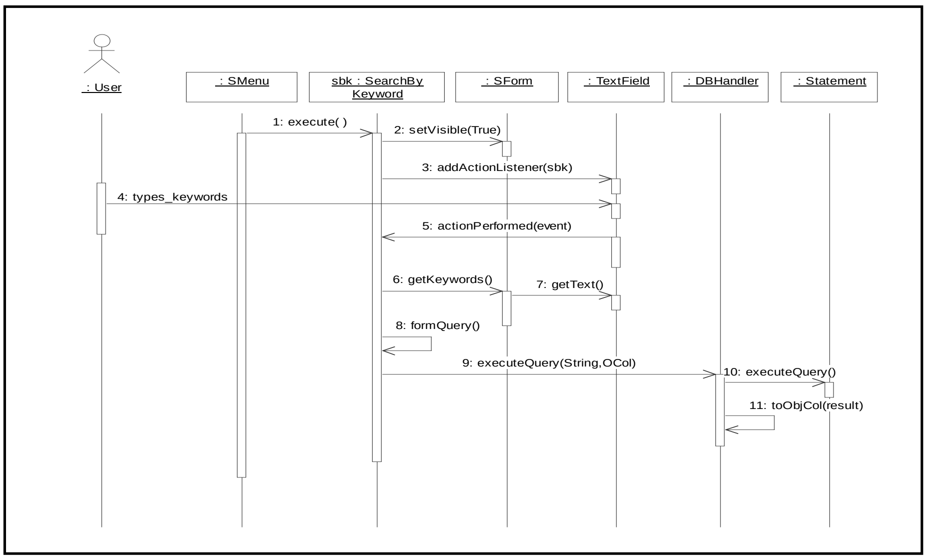

Figure 3. I_2 **-** Interaction for searching by keywords

According to Definition 2, the beliefs in the coordination capacity of the classes *SearchByKeyword*, *DBHandler*, *SMenu* in the diagram I_2 are:

(i) m_2(coord-c(SearchByKeyword,{I_2})) = 0.8

since Com(SearchByKeyword,{I_2}) = {SMenu, SForm, TextField, DBHandler} and Classes({I_2}) = {SMenu, SearchByKeyword, SForm, TextField, DBHandler, Statement}

(ii) m_2(coord-c(DBHandler,{I_2})) = 0.4

since Com(DBHandler,{I_2}) = {Statement, SearchByKeyword}) and Classes({I_2}) is as in (i) above

(iii) m_2(coord-c(SMenu,{I_2})) = 0.2

since Com(SMenu,{I_2}) = {SearchByKeyword} and Classes({I_2}) is as in (i) above

These beliefs reflect the strong coordination capacity of *SearchByKeyword* in the entire interaction, the moderate coordination capacity of *DBHandler* for only a part of the interaction and the almost negligible coordination capacity of *SMenu*.

Note however that, the above beliefs change if both diagram I_1 and diagram I_2 are taken into account. In this case, we have:

(i) m_2(coord-c(SearchByKeyword, {I_1,I_2})) = 0.571
since Com(SearchByKeyword,{I_1,I_2}) = {SMenu, SForm, TextField, DBHandler}) and Classes({I_1,I_2}) = {SMenu, SearchByKeyword, SearchByTitle, SearchByAuthor, SForm, TextField, DBHandler, Statement}

(ii) m_2(coord-c(SMenu, {I_1,I_2})) = 0.428
since Com(SMenu,{I_1,I_2}) = { SearchByKeyword, SearchByAuthor, SearchByTitle}) and Classes({I_1,I_2}) is as in (i) above

(iii) m_2(coord-c(DBHandler, {I_1,I_2})) = 0.285
since Com(DBHandler, {I_1,I_2}) = {Statement, SearchByKeyword} and Classes({I_1,I_2}) is as in (i) above

The new beliefs are affected by the high coordination capacity of the class *SMenu* in I_1, the low coordination capacity of the class *SearchByKeyword* in I_1 and the lack of any coordination capacity of the class *DBHandler* in I_1.

2.3 Functional essentiality of attributes and association ends

Attributes and association ends may provide the only channels for sending messages between the instances of the classes connected to them. Consider, for instance, an interaction where an instance of a class c_i sends a message to an instance of another class c_j. Unless c_i has an attribute or an association end whose type is the class c_j (and therefore its instances have a means of holding references to the instance of c_j) or the message has an argument of type c_j, the instance of c_i will not be able to identify and send the message to the instance of c_j.

Note also that in cases where c_i has more than one attributes or navigable association ends of type c_j it is impossible to identify from the model which of these attributes or association ends is used by the sender of the message[2]. Nevertheless, it is plausible to assume that the more the messages sent by the instances of c_i (or its subclasses) to instances of the type of an attribute or association end *a* and the fewer the other attributes or association ends of c_i having the same type as *a*, the higher the chance that at least one of these messages is dispatched through *a* and thus the higher the functional essentiality of *a* for the class c_i. Drawing upon this observation, we define the belief to the functional essentiality of attributes and association end as follows:

Definition 3: The belief to the functional essentiality of an attribute or association end *a* for a class *c* in a model *M* (denoted by the predicate *fessen-a(a,c)*) is defined as:

$m_3(\text{fessen-a}(a,c)) =$

$1 - (1-1/(|\text{Rel}(a,c)| + 1))^{|\text{Mes}(a,\, c,\, M)|}$

$m_3(\neg\text{fessen-a}(a,c)) = 1 - m_3(\text{fessen-a}(a,c))$

where

- *Mes(a,c,M)* is the set of those messages sent by the instances of *c* (or its subclasses) to instances of the type of the attribute or the association end *a* which do not have an argument of the same type as *a*
- *Rel(a,c)* is the set of the attributes and navigable association ends defined in or inherited by the class *c* that have the same type as *a*

m_3 measures the likelihood of the instances of *c* sending messages to objects that constitute the value of the attribute or association end *a*. In Definition 3, the cardinality of *Rel(a,c)* is increased by one to account for the possibility of sending the message to an instance of *c* that is created within the method that implements the operation invoked by the message. This is necessary since this creation might not be evident from the interaction itself.

According to Definition 3, the beliefs to the functional essentiality of the association end *sform* and the attribute *key* for the class *SearchByKeyword* in Figure 1 − given the sequence diagram of Figure 2 – are 0.75 and 0, respectively. These beliefs reflect the fact that *sform* is likely to be the association end used to identify the receivers of at least one of the messages in the diagram sent to instances of *SearchByKeyword*. Unlike it, the attribute *key* does not appear to have any functional role for *SForm* since no messages are sent to instances of its type (that is the class *Keyword*).

An inconsistency involving a functionally essential attribute or association end is significant because it may affect the ability of the objects to request the execution of operations.

[2] The graphical syntax of UML for sequence diagrams does not allow the specification of the exact attribute or association end whose value is used as the receiver of a message in an interaction.

2.4 Operation characterictity

An operation overridden by most of the classes in its scope, that is the set of the classes which introduce or inherit it in a model, is significant for the design of a system because it constitutes a basic kind of behaviour which must be provided by objects of different types (even if realised in different ways by these objects). We refer to this characteristic of operations as "operation characterictity" and define the belief to it as follows:

Definition 4: The belief to the characterictity of an operation *o* in a model *M* (denoted by the predicate *char-o(o)*) is defined as

$m_4(\text{char-o(o)}) =$

$\Pi_{c\ \varepsilon\ \text{Oclasses(o)}}\ |\text{Ov(o,c)}\cup\{c\}|\ /\ |\ \text{c.Sub*}\cup\{c\}|$

$m_4(\neg\text{char-o(o)}) = 1 - m_4(\text{char-o(o)})$

where

- *Oclasses(o)* is the set of the most general superclasses of the class of o which define an operation with the same signature as o
- *Ov(o,c)* is the set of the subclasses of *c* which override *o*

m_4 measures the likelihood of an arbitrary class in each of the possible scopes of an operation overriding it.

According to Definition 4, the beliefs to the characterictity of the operations *execute()* and *getImp()* in the class diagram of Figure 1 are 0.875 and 0.125, respectively. The former belief measure reflects the fact that *execute()* is an operation that has to be defined in every command class (since it is used to trigger the execution of these commands (Gamma et al., 1994)) but implemented differently by each of these command classes. Unlike it, the operation *getImp()*, which returns the object that implements a command, has a single implementation in the abstract command class *Command*. The fact that *getImp()* is not overridden by any of the different command classes in the Is-a hierarchy of Figure 1 indicates the relatively insignificant functional role of it for these classes.

2.5 Coordination capacity of messages

Messages in interactions are exchanged between objects to invoke operations in these objects. These operations may: (a) provide part of the internal functionality of the object, or (b) coordinate the interaction of a group of other objects by invoking other operations in them, combining the data that the latter operations may generate, and eventually notifying the combined outcome of the interaction to the object that invoked them.

The operations of the latter kind (and therefore the messages invoking them) are more critical for the design of the system than those of the former kind. This is because they realise the protocols of the required coordination between objects. Note, however, that in a UML design model, the only evidence about the operations invoked when a specific operation is executed comes from the messages dispatched by the message that invokes the operation. Also, depending on the elaboration stage of a model, the messages which appear in sequence diagrams may not have counterpart operations defined for the classes of their receivers (or their superclasses) in the class diagrams. To cope with these phenomena, we have defined the coordination capacity as a characteristic of messages:

Definition 5: The belief to the coordination capacity of a message *m* in a subset *S* of the interactions of a model *M* (denoted by the predicate *coord-m(m,S)*) is defined as:

$m_5(\text{coord-m}(m,S)) = |Dsig(m,S)| / |Asig(m,S)|$ *if* $Asig(m,S) \neq \varnothing$

$m_5(\text{coord-m}(m, S)) = 0$ *if* $Asig(m,S) = \varnothing$

$m_5(\neg\text{coord-m}(m,S)) = 1 - m_5(\text{coord-m}(m,S))$

where

- *Dsig(m,S)* is the set of the signatures of the messages directly dispatched by *m* in the interactions of *S*
- *Asig(m,S)* is the set of the signatures of the messages which are directly or indirectly dispatched by *m* in the interactions of *S*

m_5 measures the likelihood of an arbitrary message *x* in the transitive closure of the messages dispatched by a message *m* being directly (as opposed to indirectly) dispatched by *m*.

According to Definition 5, the beliefs to the coordination capacity of the messages *execute()*, *actionPerformed(event)* and *setVisible(True)* in the interaction of Figure 3 are:

– m_5(coord-m(execute(),{I_2})) = 1

  since

  Asig(execute(),{I_2}) = {setVisible(True), addActionListener(sbk)} and

  Dsig(execute(),{I_2}) = {setVisible(True), addActionListener(sbk)}

– m_5(coord-m(actionPerformed(event),{I_2})) = 0.5

  since

  Asig(actionPerformed(event),{I_2}) = {getKeywords(), getText(), formQuery(), executeQuery(String,OCol), executeQuery(), toObjCol(result)} and

  Dsig(actionPerformed(event),{I_2}) = {getKeywords(),formQuery(), executeQuery(String,OCol)}

– m_5(coord-m(setVisible(True),{I_2})) = 0

  since

  Asig(setVisible(True),{I_2}) = {} and

  Dsig(setVisible(True), {I_2}) = {}

These beliefs indicate that *execute()* has a co-ordination capacity in the start of the interaction I_2 where it displays the search form and registers the command class *SearchByKeyowrd* as a listener to the text field that the user may use to type in the keywords, *actionPerformed(event)* has some co-ordination capacity in the part of the interaction that executes the search, and *setVisible(True)* has no coordination capacity.

2.6 Functional dominance of messages

We consider messages that invoke operations triggering a substantial part of the behaviour of objects in an interaction as being functionally dominant in it. In our framework, the basic belief to the functional dominance of a message *m* in an interaction is defined as the likelihood of an arbitrary message in it being dispatched by *m* as shown below:

Definition 6: The belief to the functional dominance of a message *m* in a set of interactions S of a model *M* (denoted by the predicate *fdom-m(m,S)*) is defined as:

m_6(fdom-m(m,S)) = (|Asig(m,S)|+1)/|Sg(m,S)|

$m_6(\neg$fdom-m(m,S)) = 1 – m_6(fdom-m(m,S))

where

– Sg(m,S) is the set of the signatures of the messages in the interactions of set S which are sent and received by the classes (not actors) in the interactions of S excluding the signature of m.

According to Definition 6, the beliefs to the functional dominance of the messages *execute(), actionPerformed(event),* and *executeQuery(String,OCol)* in the sequence diagram I_2 are:

– m_6(fdom-m(execute(),{I_2})) = 0.22

Asig(execute(),{I_2}) = {setVisible(True), addActionListener(sbk)} and

Sg(execute(),{I_2}) = {setVisible(True), addActionListener(sbk), actionPerformed(event), getKeywords(), getText(), formQuery(), executeQuery(String,OCol), executeQuery(), toObjCol(result)}

– m_6(fdom-m(actionPerformed(event),{I_2})) = 0.66

since

Asig(actionPerformed(event),{I_2}) = {getKeywords(), getText(), formQuery(), executeQuery(String,OCol), executeQuery(), toObjCol(result)} and

Sg(actionPerformed(event),{I_2}) = {execute(), setVisible(True), addActionListener(sbk), getKeywords(), getText(), formQuery(), executeQuery(String,OCol), executeQuery(), toObjCol(result)}

– m_6(fdom-m(*executeQuery(String,OCol),*{I_2})) = 0.22

since

Asig(executeQuery(String,OCol),{I_2}) = {executeQuery(), toObjCol(result)} and

Sg(executeQuery(String,OCol),{I_2}) = {execute(), setVisible(True), addActionListener(sbk), actionPerformed(event), getKeywords(), getText(), formQuery(), executeQuery(), toObjCol(result)}

These belief measures reflect the fact that the message *actionPerformed(event)* triggers a substantial part of the entire interaction while the other two messages trigger only small parts of it.

3. Specification of consistency rules and significance criteria

As we discussed in Section 1, we define an inconsistency as a violation of a specific consistency rule. To assess the significance of inconsistencies, our framework introduces a scheme for specifying significance criteria and associating them with consistency rules. These criteria define the characteristics that the elements involved in the violation of a rule should have for the violation to be significant.

We express consistency rules using the Object Constraint Language (OCL) which is defined as part of (OMG, 1999) and significance criteria using a subset of OCL and the predicates introduced in Section 2, and wrap them in UML objects related as indicated in the extension of the UML meta-model that we have made and is shown in Figure 4.

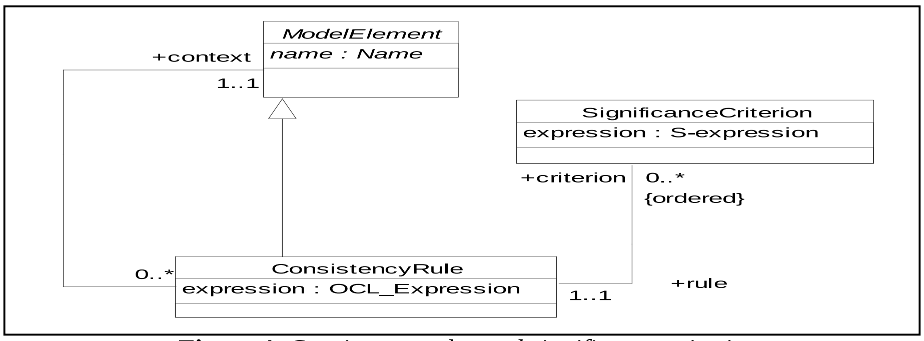

Figure 4. *Consistency rules and significance criteria*

As shown in Figure 4, each consistency rule is associated with a specific UML model element, called the "context" of the rule. Consequently, the OCL expression that specifies the rule can make references to all the named structural and behavioural features of its context as well as to the associations and generalisations which may relate it to other model elements. The classes of a UML model along with built-in OCL types which represent primitive data types and collections of values/objects (for example *Set* (OMG, 1999)) are the legitimate types for the OCL expressions written for it.

An OCL expression specifies conditions over the values of the features it references using the standard logical operators "and", "or", "implies" and "not" and the set operators "forall" and "exists". The semantics of these set operators are the same as the semantics of the universal and existential quantifier of predicate calculus. Thus, an expression of the form *set->forall(x | OCL-*

condition-over-x) and *set->exists(x | OCL-condition-over-x)* becomes true if *OCL-condition-over-x* is true for all or at least one of the elements of *set*, respectively.

As an example of specifying consistency rules using OCL consider a rule requiring that for every message in a sequence diagram there must be either an association or an attribute between its sender and its receiver navigable from the former to the latter class. This rule can be defined in the context of the UML meta-class *Message* (i.e., the class of all the messages which appear in the interactions of a model, see Figure 5) using OCL as follows[3]:

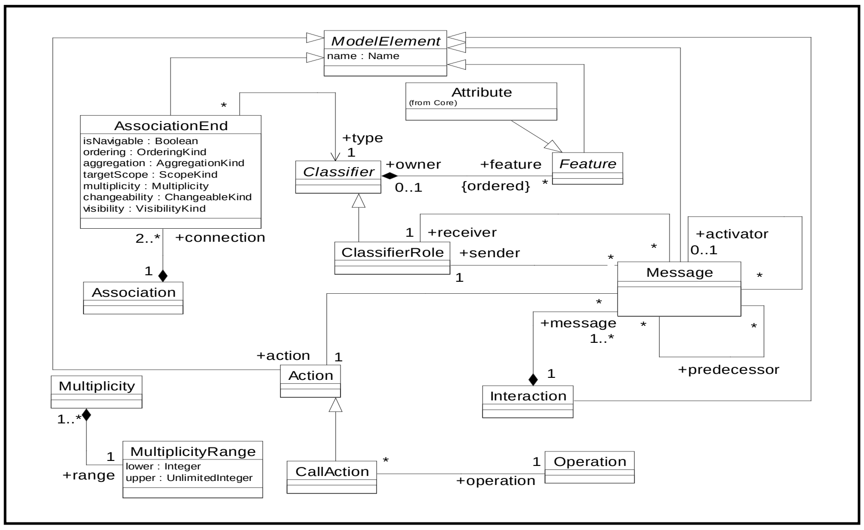

Figure 5. *UML model elements (adopted from (OMG, 1999))*

Rule 1

context: Message

expression:

self.*action*.**oclIsTypeOf**(*CallAction*) **implies self**.*sender.feature*–>**exists**(a |

a.**oclIsTypeOf**(*Attribute*) **and**

(a.*type* = **self**.*receiver*) **or** *Association*.**allInstances**–>**exists**(r | r.*connection*–>**exists**(e_1, e_2 |

(e_1 <> e_2) **and** (e_1.*type* = **self**.*sender*) **and** (e_2.*type* = **self**.*receiver*) **and** (e_2.*isNavigable* = **True**)))

[3] In OCL and S-expressions strings in boldface and Italics are reserved OCL keywords and names established in the UML meta-model, respectively. The keyword **self** in these expressions refers to an instance of the class that constitutes the context of the consistency rule and consequently the context of the S-expression that defines a criterion associated with it.

A significance criterion in our framework is specified by a significance expression (*S-expression*) and must be associated with a consistency rule (see Figure 4). The S-expression specifies a logical combination of the characteristics which the model elements giving rise to the violation of the rule (or other model elements connected to them) are required to have for the inconsistency to be significant. These characteristics are specified by using the special predicates defined in Section 2. An S-expression has the same context as the consistency rule associated with the criterion it defines and, therefore, it can reference any named feature in the closure of the features of the model elements which are reachable from this context.

Atomic S-expression	**Belief**	**Type validity condition**
gen-c(elem)	Bel(gen-c(elem)) = m_1(gen-c(elem))	elem.*type* = *Class*
fessen-a($elem_1$,$elem_2$)	Bel(fessen-a($elem_1$,$elem_2$)) = m_3(fessen-a($elem_1$,$elem_2$))	$elem_1$.*type* = *Attribute* OR $elem_1$.*type* = *AssociationEnd* AND $elem_2$.*type* = *Class*
char-o(elem)	Bel(char-o(elem)) = m_4(char-o(elem))	elem.*type* = *Operation*
coord-c($elem_1$,$elem_2$)	Bel(coord-c($elem_1$,$elem_2$)) = m_2(coord-c($elem_1$,$elem_2$))	$elem_1$.*type* = *Class* AND $elem_2$.*type* = **Set** (*Interaction)*
coord-m($elem_1$,$elem_2$)	Bel(coord-m($elem_1$,$elem_2$)) = m_5(coord-m($elem_1$,$elem_2$))	$elem_1$.*type* = *Message* AND $elem_2$.*type* = **Set** (*Interaction)*
fdom-m($elem_1$,$elem_2$)	Bel(fdom-m($elem_1$, $elem_2$)) = m_6(fdom-m($elem_1$, $elem_2$))	$elem_1$.*type* = *Message* AND $elem_2$.*type* = **Set** (*Interaction)*

Table 1. *Syntactic forms, typing conditions and beliefs for valid atomic S-Expressions*

Tables 1 and 2 present the syntactic forms of the S-expressions definable in our framework and the typing conditions that these expressions have to satisfy in order to be valid. More specifically, Table 1 presents the syntactic forms of, and the type validity conditions for the so-called "atomic S-expressions" (these are expressions consisting of only one of the predicates introduced in Section 2). The type validity condition determines the valid type(s) for the element(s) that the predicate of an expression refers to. Table 2 presents the syntactic forms of, and the validity conditions for "non atomic S-expressions" (these are logical combinations of atomic

S-expressions). Thus, for instance, according to Table 1 the S-expression *gen-c(elem)* is valid only if the type of the model element denoted by *elem* is the UML meta-class *Class*. The complete grammar for S-expressions is given in (Spanoudakis, 1999).

As an example of specifying a significance criterion consider the case where the violations of *Rule-1* above should be considered significant only if they are caused by messages which are functionally dominant and have coordinating capacity in their interactions. This criterion of significance can be specified as follows:

Criterion

Rule: Rule-1

S-expression:

fdom-m(**self**, **self.***interaction*)

and coord-m(**self**, **self.***interaction*)

In the S-expression of this criterion, "self" refers to the instances of the context of *Rule-1*, that is the UML meta-class *Message*. By using the special predicates *fdom-m* and *coord-m*, this S-expression specifies that the message that violates the rule must be functionally dominant and have a coordinating capacity in the interaction (sequence diagram) it belongs to (that is the value of the feature: **self.***interaction*).

Non atomic S-Expression		**Belief**	**Validity condition**
Non quantified expressions	p_1 **and**…**and** p_n	Bel($\text{and}_{i=1,\ldots,n}\ p_i$) = $\Pi_{i=1,\ldots,n}$ Bel (p_i)	p_i : valid atomic S-expression (forall i=1,…,n)
	p_1 **or** …**or** p_n	Bel($\text{or}_{i=1,\ldots,n}\ p_i$) = $\Sigma_{J\subseteq\{1,\ldots,n\}}(-1)^{\lvert J\rvert +1}$ Bel($\text{and}_{i\varepsilon J} p_i$)	p_i : valid atomic S-expression (forall i=1,…,n)
Quantified expressions	elem->**exists**(x | OCL-exp-over-x **and** se(x))	$\Sigma_{J\subseteq S}(-1)^{\lvert J\rvert + 1}$Bel($\text{and}_{x\varepsilon J}$ se(x)) where S = {x | (x ε elem) and OCL-exp-over-x = True}	elem.*type* = **Set**(*ModelElement*) AND se(x): is a valid non quantified S-expression over x
	elem->**forall**(x | OCL-exp-over-x **and** se(x))	$\Pi_{x\ \varepsilon\ \text{elem}}$ Bel(se(x)) If elem->**forall**(x | OCL-exp-over-x) = True) 0 If elem->**forall**(x | OCL-exp-over-x) = False)	elem.*type* = **Set**(*ModelElement*) AND se(x): is a valid non quantified S-expression over x

Table 2. *Syntactic forms of and validity conditions for non atomic S-Expression*

To assess the significance of the violations of a specific consistency rule, we compute degrees of belief for the satisfiability of the S-expression of the criterion associated with the rule by the elements of the model which this expression refers to. These elements are related to the model elements that gave rise to the violation of the rule as specified by the S-expression. Subsequently, the violations of the rule are ranked in descending order of these degrees of belief.

Tables 1 and 2 show the formulas used to compute the degrees of belief for the different forms of atomic and non-atomic S-expressions. These formulas are derived using the axioms of the Dempster-Shafer theory of evidence (Shafer, 1975) as we prove in (Spanoudakis, 1999) Their derivation is based on the fact that – as we have also proven in (Spanoudakis, 1999) – the belief functions introduced in Section 2 satisfy the axiomatic foundation of Dempster-Shafer *basic probability assignments* (Shafer , 1975).

In the following section, we give an example of computing degrees of belief for the satisfiability of significance criteria and ranking inconsistencies according to them.

4. Example

As an example of detecting and assessing the significance of inconsistencies in our framework, consider the UML model consisting of the class and sequence diagram shown in Figures 1 and 3, respectively. These diagrams are inconsistent with respect to *Rule-1* in Section 3 since there are no attributes and/or associations between the sender and the receiver of the following messages: *execute(), getText(), executeQuery(String,OCol).*

If the significance of these inconsistencies is assessed according to *Criterion-1* in Section 3, the inconsistencies caused by the messages *execute()* and *executeQuery(String,OCol)* become the ones with the highest significance, followed by the inconsistency caused by the message *getText()*. This is because the degrees of belief about the satisfiability of *Criterion-1* by each of these messages are (according to the belief functions of Tables 1 and 2):

1) Bel(fdom-m(execute(),{I_2}) **and** coord-m(execute(),{I_2})) =
   m_6(fdom-m(execute(),{I_2}) $\times$ m_5(coord-m(execute(),{I_2})) = 0.22 $\times$ 1 = 0.22

2) Bel(fdom-m(executeQuery(String,OCol),{I_2}) **and** coord-m(executeQuery(String,OCol),{I_2})) = m_6(fdom-m(executeQuery(String,OCol), {I_2})) $\times$ m_5(coord-m(executeQuery(String,OCol),{I_2})) = $0.22 \times 1 = 0.22$
3) Bel(fdom-m(getText(),{I_2}) **and** coord-m(getText(),{I_2})) = m_6(fdom-m(getText(),{I_2})) $\times$ m_5(coord-m(getText(),{I_2})) = $0 \times 0 = 0$

5. Experimental evaluation

To evaluate our framework, we implemented the belief functions defined in Section 2 using the scripting language of the CASE tool Rational Rose (Rational, 1998) (a tool supports UML) and conducted a series of preliminary experiments using this implementation.

	MODEL														
Size	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
No. of Classes	19	26	23	14	25	18	25	40	12	16	74	13	43	39	55
No. of Seq. Diagrams	3	4	4	2	4	3	12	9	8	8	3	3	12	13	6
No. of Messages	68	24	32	43	51	44	155	92	87	115	65	36	133	186	185
No. of Associations	82	91	68	29	31	38	79	60	17	17	187	30	198	97	230
Producer	a	a	a	a	a	a	a	a	a	a	a	a	a	b	a
Legend:	a	MSc student													
	b	Group of MSc students													

Table 3. *Size of models used in experimental evaluation*

Our experiments were aimed at testing:

a) whether the satisfiability measures calculated for significance criteria definable in the framework are of sufficient diversity for producing elaborate rankings of inconsistencies, and
b) whether the rankings of significance produced by the criteria of our framework are compliant with rankings of significance produced by those who developed the models and/or other expert developers when the same criteria are taken into account.

In these experiments, we used 15 UML models produced by postgraduate students doing an MSc course in Object-Oriented Software Systems in the Department of Computing at City

University. Measures of the size of each of these models are shown in Table 3. The findings of these experiments are discussed below.

5.1 First Set of Experiments: Design, Results and Discussion

The objective of our first set of experiments was to establish whether the criteria definable in our framework can produce elaborate rankings of significance of inconsistencies. To conduct this experiment, we checked each of the different models against three consistency rules:

(i) The consistency rule *Rule-1* defined in Section 3. Recall that this rule requires that for every message in an interaction there must be either an association or an attribute between the class of its sender and the class of its receiver navigable from the former to the latter class.

(ii) A consistency rule requiring that the class of the receiver of a message in an interaction defines or inherits an operation with the same signature as the message. This rule is defined in OCL as follows:

Rule-2

context: *Message*

S-expression: **self**.*action*.**oclIsTypeOf**(*CallAction*) **implies** **self**.*receiver.feature*–>**exists**(o:*Operation* (**self**.*action*.*operation* = o)**)**

(iii) A consistency rule requiring that the lower multiplicity bound of an association end that is attached to a class whose instances receive at least one message from instances of the class attached to the other end of its association must be greater or equal to 1. This rule is specified in OCL as follows:

Rule-3

context: *AssociationEnd*

S-expression: **self**.*association*–>**exists**(*a*:*Association*| *a.connection*–>**exists**(e_1, e_2 | (e_1 = **self)** **and** (e_1 <> e_2) **and** (e_1.*type* = c_1) **and** (c_1.**oclIsTypeOf**(*Classifier*)) **and** (e_2.*type* = c_2) **and** (c_2.**oclIsTypeOf**(*Classifier*)) **and** (c_2.message–>**exists**(m: *Message* | *m.receiver* = c_1)))) **implies** (**self.***mutliplicity.range.lower* >= 1)

Criterion	**S-Expression**	**Meaning**
Criterion 1	fdom-m(**self**, **self**.*interaction*)	The message has functional dominance in the sequence diagram it appears.
Criterion 2	coord-m(**self**, **self**.*interaction*)	The message has a co-ordinating capacity in the sequence diagram it appears.
Criterion 3	coord-c(**self.***receiver*, **self**.*interaction*)	The receiver class of a message has a co-ordinating capacity in the specific sequence diagram that includes the message.
Criterion 4	coord-c(**self.***sender*, **self**.*interaction*)	The sender class of a message has a co-ordinating capacity in the specific sequence diagram that includes the message.
Criterion 5	coord-m(**self**, **self**.*interaction*) **or** coord-c(**self.***receiver*,**self**.*interaction*) **or** coord-c(**self.***sender*, **self**.*interaction*)	The message or its receiver class or its sender class has a co-ordinating capacity in the specific sequence diagram.
Criterion 6	fessen-a(**self**, **self**.*association.oppositeend.type*)	The association end is functionally essential for the class attached to the other end of its association.

Table 4. *Criteria used to assess the significance of the violations of Rule 1 and 2.*

The significance of the violations of Rule-1 was assessed using the criteria 1 and 2 in Table 4. The significance of the violations of Rule-2 was assessed using the criteria 2, 3, 4 and 5 in Table 4. The significance of the violations of Rule-3 was assessed using the criterion 6 in Table 4.

Belief measures for the satisfiability of each of these criteria by the inconsistencies in the different models were computed and used to rank these inconsistencies as we did with the inconsistencies in the example of Section 4. More specifically, the inconsistencies caused by elements believed to satisfy a criterion to the same extent (i.e. giving rise to equal belief measures) were classified in the same category. The different categories of inconsistencies were then ranked in descending order of the criterion satisfiability beliefs computed for their elements.

Tables 5, 6 and 7 present statistics of the belief measures computed for the satisfiability of the criteria by the model elements violating the rules. The columns of each of these tables indicate the different models used in the experiments. The rows are grouped under the different criteria used to assess the significance of the inconsistencies. The tables show the number of the inconsistencies detected with respect to the rule in each model (see row N_{inc}) and, for each criterion: (1) the number of the different categories of significance generated by the criterion (see rows N_c), (2) the completeness ratio of the ranking generated by the criterion RC = N_c/N_{inc} (see rows RC), (3) the mean value of the beliefs for the satisfiability of the criterion (see rows M_b), (4) the median value of the beliefs for the satisfiability of the criterion (see rows $Median_b$), (5) the standard deviation of the beliefs for the satisfiability of the criterion (see rows s_b), (6) the standard deviation of the number of inconsistencies in each category of the ranking (see rows s_{ic}), and (7) the relative variability of the beliefs for the satisfiability of the criterion (see rows s_b/m_b).

Rule 1	**MODEL**														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
N_{inc}	20	2	7	5	34	19	82	39	32	16	18	6	49	111	41
Criterion 1															
N_c	5	2	4	4	9	6	21	11	16	12	7	2	10	30	13
RC	0.25	1	0.57	0.8	0.26	0.32	0.26	0.28	0.5	0.75	0.39	0.33	0.20	0.27	0.32
M_b	0.06	0.15	0.19	0.1	0.2	0.11	0.08	0.4	0.17	0.28	0.07	0.02	0.05	0.11	0.07
$Median_b$	0.04	0.15	0.17	0.05	0.08	0.08	0.04	0.38	0.09	0.18	0.04	0	0	0	0.03
s_b	0.04	0.07	0.1	0.09	0.29	0.13	0.04	0.29	0.14	0.26	0.06	0.04	0.12	0.23	0.1
s_{ic}	3.08	0	0.5	0.5	2.99	2.56	4.32	2.46	1.55	0.49	2.44	2.82	10.27	13.66	2.82
s_b / M_b	0.64	0.47	0.54	0.88	1.46	1.14	0.53	0.73	0.81	0.93	0.92	2	2.4	2.09	1.43
Criterion 2															
N_c	2	1	2	3	6	2	6	9	8	5	4	2	6	11	5
RC	0.1	0.5	0.29	0.6	0.18	0.11	0.07	0.23	0.25	0.31	0.22	0.33	0.12	0.1	0.12
M_b	0.1	0	0.29	0.25	0.2	0.26	0.24	0.49	0.29	0.53	0.19	0.17	0.27	0.27	0.7
$Median_b$	0	0	0	0	0	0	0	0.5	0	0.5	0	0	0	0	1
s_b	0.31	0	0.49	0.43	0.37	0.45	0.38	0.39	0.39	0.44	0.34	0.4	0.44	0.43	0.41
s_{ic}	11.3	0	2.12	1.15	9.54	6.36	20.8	4.66	5.81	2.28	5.69	2.82	13.27	22.72	9.31
s_b / M_b	3.08	0	1.71	1.73	1.9	1.72	1.59	0.8	1.34	0.82	1.82	2.35	1.63	1.59	0.58

Table 5. *Statistics for the rankings of the violations of Rule-1*

The main statistic to look at in Tables 5-7 is the ranking completeness ratio (RC). When this ratio is 1 then the criterion used can fully order the inconsistencies detected. The mean value of the RC-ratios in the experiments that we conducted was M_{RC}= 0.40 or, equivalently, the criteria used produced distinct significance categories with 2.5 (= $1/M_{RC}$) inconsistencies in each category on average. Thus, it may be argued that on average the criteria used in our first set of experiments were capable of producing elaborate rankings of significance.

It has also to be appreciated that the above mean value of the RC-ratios resulted from a set of experiments in which 5 out of the 6 significance criteria used were atomic S-expressions (Criteria 1, 2, 3, 4 and 6) concerned with single characteristics of model elements. Evidently from the statistics for Criterion 5 in Table 6, in the only case where we used a significance criterion referring to a logical combination of characteristics of elements giving rise to inconsistencies, the resulted RC-ratios were significantly higher: the mean value of the RC-ratios for Criterion 5 was 0.54.

Rule 2	**MODEL**														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
N_{inc}	8	11	1	7	26	35	90	39	7	14	65	0	16	30	131
Criterion 2															
N_c	1	4	1	3	9	4	9	4	7	3	14	0	8	11	13
RC	0.13	0.36	1	0.43	0.35	0.11	0.1	0.1	1	0.21	0.22	0	0.5	0.37	0.1
M_b	0	0.24	0	0.44	0.17	0.3	0.11	0.02	0.41	0.13	0.33	0	0.57	0.21	0.39
Median$_b$	0	0	0	0.09	0	0	0	0	0.38	0	0.11	0	0.75	0.17	0.4
s_b	0	0.37	0	0.52	0.32	0.45	0.26	0.05	0.3	0.33	0.4	0	0.46	0.17	0.41
s_{ic}	0	2.87	0	1.15	4.94	10.4	23	16.8	0	6.35	7.99	0	2.45	2.05	16.9
s_b / M_b	0	1.52	0	1.19	1.86	1.49	2.34	3.22	0.73	2.56	1.2	0	0.81	0.81	1.05
Criterion 3															
N_c	3	3	1	3	11	7	15	6	5	5	9	0	6	8	14
RC	0.38	0.27	1	0.43	0.42	0.2	0.17	0.15	0.71	0.36	0.14	0	0.37	0.27	0.11
M_b	0.17	0.26	0.29	0.21	0.25	0.29	0.21	0.28	0.39	0.27	0.22	0	0.42	0.38	0.28
Median$_b$	0.11	0.17	0.29	0.25	0.18	0.29	0.15	0.25	0.42	0.29	0.18	0	0.43	0.37	0.25
s_b	0.13	0.16	0	0.07	0.19	0.16	0.16	0.2	0.08	0.13	0.12	0	0.10	0.1	0.21
s_{ic}	2.08	1.71	0	2.31	1.75	3.11	5.63	5.75	0.55	1.92	5.45	0	3.14	2.53	11.4
s_b / M_b	0.72	0.6	0	0.31	0.73	0.54	0.77	0.73	0.22	0.5	0.53	0	0.24	0.26	0.75
Criterion 4															
N_c	3	3	1	3	11	8	17	8	5	7	8	0	6	8	13
RC	0.38	0.27	1	0.43	0.42	0.23	0.19	0.21	0.71	0.5	0.12	0	0.37	0.27	0.1
M_b	0.33	0.29	0.14	0.27	0.48	0.47	0.42	0.57	0.2	0.6	0.31	0	0.22	0.22	0.46
Median$_b$	0.44	0.25	0.14	0.25	0.56	0.57	0.46	0.63	0.08	0.58	0.27	0	0.17	0.14	0.5
s_b	0.16	0.14	0	0.15	0.22	0.18	0.2	0.17	0.26	0.25	0.15	0	0.10	0.2	0.23
s_{ic}	2.08	1.53	0	2.31	1.91	3.93	4.95	6.27	0.55	1.91	4.12	0	2.65	2.76	8.16
s_b / M_b	0.47	0.49	0	0.55	0.46	0.39	0.46	0.29	1.29	0.41	0.47	0	0.45	0.91	0.5
Criterion 5															
N_c	4	8	1	4	17	14	36	11	7	8	33	0	9	22	31
RC	0.5	0.73	1	0.57	0.65	0.4	0.4	0.28	1	0.57	0.51	0	0.56	0.73	0.24
M_b	0.46	0.6	0.39	0.69	0.68	0.72	0.58	0.69	0.75	0.77	0.63	0	0.73	0.5	0.53
Median$_b$	0.51	0.63	0.39	0.62	0.72	0.72	0.59	0.72	0.68	0.88	0.59	0	0.9	0.48	0.62
s_b	0.19	0.25	0	0.31	0.2	0.22	0.22	0.15	0.14	0.18	0.25	0	0.3	0.17	0.36
s_{ic}	0.82	0.52	0	0.96	1.07	2.41	2.24	4.2	0	1.04	2.36	0	2.33	0.79	7.2
s_b / M_b	0.4	0.42	0	0.45	0.28	0.3	0.37	0.22	0.19	0.24	0.39	0	0.41	0.34	0.68

Table 6. *Statistics for the rankings of the violations of Rule-2*

An analysis of the RC-ratios shown in Tables 5, 6 and 7 showed that their distribution had a positive skewness (degree of skewness = 0.93, $\text{Median}_{RC} = 0.35 < M_{RC}$). This distribution is shown in Figure 6. Also the standard deviation of the RC-values was: s_{RC} = 0.27. These statistics indicate that further experimentation is needed to confirm the argument about the power of the framework to produce elaborate rankings of inconsistencies. This is because in 50 per cent of the cases the RC-ratio was lower than 0.35 and there was a relative high deviation of these values from their average.

Rule 3	**MODEL**														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
N_{inc}	1	10	5	9	1	2	10	8	12	10	14	7	23	7	19
Criterion 6															
N_c	1	3	5	4	1	2	5	3	6	3	5	4	8	3	8
RC	1	0.33	0.66	0.44	1	1	0.5	0.37	0.5	0.33	0.36	0.57	0.35	0.42	0.42
M_b	0.99	0.41	0.55	0.63	0.5	0.93	0.55	0.62	0.70	0.65	0.64	0.45	0.70	0.66	0.61
$Median_b$	0.99	0.44	0.55	0.5	0.5	0.93	0.58	0.5	0.70	0.75	0.5	0.5	0.75	0.75	0.75
s_b	0	0.11	0.26	0.18	0	0.08	0.18	0.19	0.20	0.24	0.20	0.35	0.23	0.16	0.37
s_{ic}	0	0.58	0.57	1.5	0	0	1	2.08	1.55	4.04	2.38	0.95	2.36	1.15	1.5
s_b / M_b	0	0.27	0.47	0.29	0	0.09	0.33	0.31	0.28	0.37	0.31	0.78	0.33	0.24	0.61

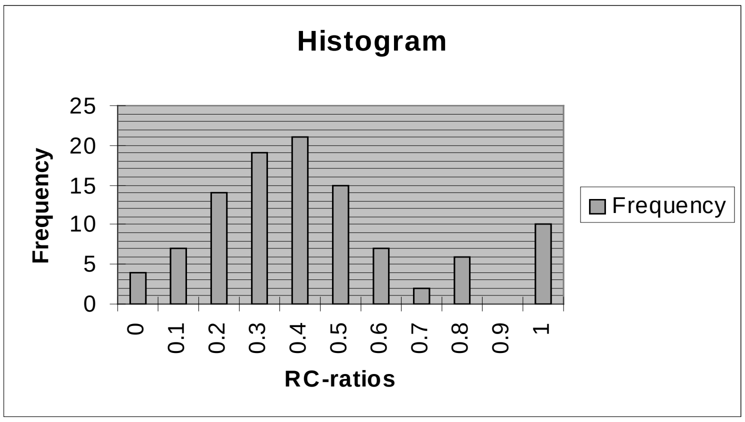

Table 7. *Statistics for the rankings of the violations of Rule-3*

Figure 6. *Distribution of RC-ratio values.*

To explore further the differences in the RC-ratio values, we analysed the correlation of these values with: (a) different measures of the size of the models (in particular, the number of classes ($N_{Classes}$) and messages ($N_{Messages}$)), (b) the number of the inconsistencies detected in each case (N_{inc}), and (c) the relative variability of the belief measures computed for the satisfiability of the criteria (i.e., S_b/M_b).

The correlation coefficients calculated by this analysis are shown in Table 8. The most prominent result of this analysis was that the RC-ratios were found to be negatively correlated with the number of the inconsistencies (N_{inc}); the correlation coefficient between these measures and the RC-ratios was −0.46, as shown in Table 8.

	$N_{Classes}$	$N_{Messages}$	N_{inc}	S_b/M_b	RC
$N_{Classes}$	1				
$N_{Messages}$		1			
N_{inc}	0.48	0.62	1		
S_b/M_b	0.05	0.11		1	
RC	-0.27	-0.26	-0.46	-0.37	1

Table 8. *Correlation of RC-ratios with model size, inconsistency and belief variability measures*

Our explanation of this phenomenon is the following. The number of the inconsistencies (N_{inc}) was found to be positively correlated with the model size: its correlation coefficient with the number of model classes ($N_{classes}$) was 0.48 and its correlation coefficient with the number of model messages ($N_{messages}$) was 0.62. This was expected as larger models are more likely to breach consistency rules. What we were also expecting in the case of larger models, however, was that the variability of the belief measures computed for the satisfiability of the used significance criteria (i.e., the ratios S_b/M_b) would also be higher. And, higher variability of the belief measures was expected to lead to finer grain distinctions in the significance rankings. This expectation did not turn out to be correct. As shown in Table 8, the S_b/M_b ratio had only a very weak positive correlation with the size of the model: its correlation coefficient with the number of model classes ($N_{classes}$) was 0.05 and its correlation coefficient with the number of model messages ($N_{messages}$) was 0.11. The weak correlation between S_b/M_b and the different measures of the size of the models is explained by the fact that the criteria used in our experiments were concerned with only specific parts of a model (e.g. specific interactions) and, therefore, they were not affected by the overall size of the models. Clearly, further experimentation is needed to explore under what circumstances higher RC-ratios can be obtained.

5.2 Second Set of Experiments: Design, Results and Discussion

In the second set of experiments, we tried to establish whether the rankings of inconsistencies produced by the criteria definable in the framework: (a) preserve the order of rankings of significance produced by humans, and (b) are as elaborate as the latter rankings. To conduct this set of experiments, we selected randomly:

- 4 of the models used in the first set of experiments, and
- 12 different inconsistencies detected with respect to Rule 1 (see Section 3) in each of these models

A ranking of the inconsistencies in each model was produced by the degrees of belief to the satisfiability of the following criterion (*framework ranking of significance*):

Criterion-7

Rule:	Rule-2
S-expression:	(fdom-m(**self**, **self.***interaction*) **and** coord-m(**self**, **self.***interaction*)) **or** coord-c(**self**, **self**.*receiver*)

Criterion 7 was used to spot as significant inconsistencies caused by messages which had a co-ordinating capacity and were functionally dominant in an interaction or which were invoking operations in classes with a co-ordinating capacity in it.

Subsequently, the author of each model was asked to indicate the significance of the same inconsistencies in his/her model on the scale 1-10, with "1" denoting that the inconsistency had no significance at all and "10" denoting that the inconsistency was very significant. The authors were prompted to use Criterion 7. The significance scores given by the authors were used to rank the inconsistencies of each model (*author ranking of significance*). The same models and inconsistencies were also given to an expert who have had a Ph.D. in Computer Science and 8 years of experience in object-oriented modelling. The expert was also asked to indicate the significance of inconsistencies using the same scale and following the instructions given to the authors. The significance scores given by the expert produced a third ranking of significance of the inconsistencies in each model (*expert ranking of significance*).

The author and expert rankings of the inconsistencies in each model were correlated with the corresponding system rankings using the Kendall's *tau* (τ) coefficient for rankings with ties (Hays, 1969). Kendall's τ coefficient is defined as the difference between the probability of two rankings agreeing about a pair and the probability of two rankings disagreeing about a pair

$$\tau = \text{Pr (two rankings agree about a pair)} - \text{Pr (two rankings disagree about a pair)}$$

Table 9 shows the main statistics computed from the correlation of the rankings. More specifically, it shows:

- the τ coefficients for the correlation of the author ranking with the system ranking (τ_a)

and the correlation of the expert ranking with the system ranking (τ_e)

- the values of the statistic S (i.e. the number of agreements minus the number of disagreements of two rankings) used in the calculation of τ for the correlation of the author ranking with the system ranking (S_a) and the correlation of the expert ranking with the system ranking (S_e), and
- the values of the ratio $z = S/\sigma_s$ (σ_s is an estimate of the standard deviation of the values of the statistic S calculated as described in the appendix of the paper) for both correlations

In Appendix 1, we give the formulas for and an example of calculating these statistics.

Statistics	MODEL			
	7	8	9	11
τ_a	-0.284	-0.253	0.174	0.53
S_a	-10	-14	7	32
z_a	-1.08	-1.03	0.65	2.10
τ_e	0.112	0.279	0.098	0.51
S_e	6	14	4	25
z_e	0.447	1.1	0.371	1.96

Table 9. *Kendall's Rank Correlations Statistics*

As shown in Table 9, the rankings produced by the assessments of significance given by the authors had a negative correlation with the rankings produced by the system for two of the models (models 7 and 8) and a positive correlation for the other two models (models 9 and 11). The rankings produced by the assessments of significance given by the expert had a positive correlation with the rankings produced by the system for all the four models. Note, however, that only in the case of model 11 the detected positive correlations were found to be statistically significant (a = 0.05). The test for the statistical significance of the correlations was based on the z ratio which is known to have a normal distribution[4].

	MODEL			
	7	8	9	11
RC_f	0.66	0.66	0.75	0.42
RC_a	0.25	0.33	0.25	0.42
RC_e	0.33	0.25	0.27	0.25

Table 10. *RC-ratios of human- and framework-based rankings of significance*

[4] For rankings with ties, the z-ratio has the normal distribution only if the two rankings show the same distribution of ties.

Our second set of experiments also showed that the rankings of significance produced by the human subjects were not as elaborate as those produced by the framework. This can be observed from the completeness (RC) ratios of the different rankings which are shown in Table 10. The completeness ratios in this table also show that the statistically significant positive correlations between the framework and the subject rankings emerged only in the case of model 11 where the completeness ratio of the framework ranking (RC_f) was relatively closer to the completeness ratio of the rankings produced by the human subjects (RC_a and RC_e).

The second set of experiments has shown that the framework tends to create more elaborate rankings of significance than developers and that the rankings it creates are not always in agreement with rankings of significance produced by developers. These results, however, are by no means conclusive and need to be confirmed by further experimentation.

6. Related work

Work related to the framework discussed in this paper falls into two broad strands of research in the field of software engineering. The first of these strands is concerned with the problem of managing inconsistencies in software models. The second strand includes research work on software metrics.

6.1 Related work on inconsistency management

A substantial body of research has been concerned with the problem of detecting and resolving inconsistencies between software system specifications (Emmerich et al., 1999; Finkelstein et al., 1994; Hunter and Nuseibeh 1998; Spanoudakis and Finkelstein 1996; Schwanke and Kaiser 1988; Lamsweerde et al., 1998; Heitmeyer et al., 1995; Robinson and Fickas 1994; Robinson and Pawlowski 1999). Those interested may find a survey of the field in (Spanoudakis and Zisman, 2001). However, only few strands of work in this general area have been concerned with the particular problem of diagnosing the significance of inconsistencies.

Emmerich et al (1999) have developed a framework for managing the compliance of software documentation artifacts with consistency rules which realise document representation standards. In their framework, software designers can write scripts to implement diagnostic checks to assess the importance and the difficulty of making a document compliant with the rule it violates.

Hunter and Nuseibeh (1998) treat diagnosis as the identification of the "possible sources" of an inconsistency. In their work, this source is defined in terms of a set Δ of all the formulas in a software model and a subset P of Δ which contains the formulas used in the proof of an inconsistency (i.e. the derivation of the empty clause ($\perp$) from the formulas in Δ). More specifically, the possible source of an inconsistency is defined as any subset S of P whose formulas belong to Δ and for which the set of formulas $P - S$ is a set of consistent formulas. Their work supports the identification of the set S.

Robinson and Pawlowski (1998) suggest the use of two simple measures as estimates of the impact of conflicting requirement statements, namely the requirement "contention" and "average potential conflict". The contention of a requirement statement in their DealScribe system is computed as the ratio of the number of the very conflicting or conflicting relations over the total number of relations that this statement has with other requirements statements. The average potential conflict of a statement is measured as the average of the subjective probabilities of conflict that have been associated with all the conflicting and very conflicting relations that have been asserted for it. Robinson and Pawlowski (1998) claim that the contention measure has been found to be very effective in ranking conflicting requirements in terms of significance and attempting their resolution in the derived order.

Kotonya and Sommerville (1996) in their VORD method also expect the stakeholders to provide weights that indicate the order of importance of their requirements models. These weights are subsequently used to establish the importance of conflicts between these requirements.

6.2 Related work on software metrics

Software metrics similar to some of the metrics defined in our framework have been proposed in the literature but have not been used to assess the significance of inconsistencies in software models.

More specifically, the *depth of inheritance tree (DIT)* (Chidamber and Kemerer, 1994) and the *class hierarchy nesting level* (Lorenz, 1993) are similar to m_1. Note, however, that unlike m_1, DIT treats as generic classes which have no subclasses. Such classes are not as important as classes that m_1 would spot as generic since they have no subclasses that could be affected by inconsistencies involving them.

Class coupling (CBO) (Chidamber and Kemerer, 1994) and the *number of collaborating classes (NCC)* (Jacobson et al., 1995) are similar to m_2. The difference between m_2 and CBO and NCC is that m_2 provides a relative measure of inter-class collaboration in a specific set of system interactions.

An idea similar to that underpinning the definition of the functional essentiality of attributes and association ends (belief function m_3) underpins the construction of the so-called "method-to-variable connection matrix" (MVCM) in (Tegarden et al., 1995) which is used to record (and subsequently count) references to object variables by specific methods. The main difference between m_3 and the MVCM is that m_3 is not applicable to attributes with primitive values (as opposed to object-values), and that m_3 establishes the potential of using an attribute/association end to identify the receiver of a message dispatched in a method as opposed to using its value in any possible way in a method.

Finally, it has to be appreciated that what clearly differentiates the metrics used in our framework from the above software metrics is their common underlying axiomatic interpretation as D−S beliefs. This, as discussed in (Spanoudakis, 1999), provides a sound basis for deriving the beliefs for the significance criteria presented in Section 3.

7. Summary and future work

In this paper, we presented a framework for assessing the significance of inconsistencies in design models of software systems expressed in UML. This assessment is based on criteria that software designers can specify to establish combinations of characteristics that the model elements which are involved in an inconsistency should have for the inconsistency to be significant. The framework offers a predefined set of such characteristics which are indicative of the impact that an inconsistency that involves a particular model element may have for the model as a whole (see for example the characteristics of class genericity and operation charactericity) or selected parts of it (see for example the characteristics of class and message coordinating capacity).

We have also presented results of a series of experiments conducted to evaluate the framework. The main observations from these experiments were that: (a) it is possible to define criteria in the framework that generate elaborate rankings of significance of inconsistencies, (b) the rankings which are generated by the framework based on particular criteria tend to be more elaborate than rankings of significance that developers generate when prompted to use the same criteria, and (c) the rankings of significance produced by the framework tend to have a positive correlation with rankings of significance produced by experienced developers and a negative correlation with rankings of significance produced by less experienced developers.

An important issue which relates to the use of the presented framework is how to use the rankings of significance produced by it in making decisions about the handling the inconsistencies. This issue is addressed by the Reconciliation method in the context of which the diagnostic framework has been developed. Reconciliation incorporates an explicit model of the process of managing inconsistencies which defines: (a) the circumstances under which the significance of inconsistencies which arise as violations of particular consistency rules may be assessed, and (b) alternative ways of handling inconsistencies depending on their significance. Developers can specify this process model in a way that tailors the inconsistency management process to the needs of specific software development projects. They may, for instance, define this process model so as to allow inconsistencies whose significance is below a preset value to be temporarily unresolved and to require inconsistencies whose significance exceeds another preset value to be fully resolved. The process model may also be specified so as to require the handling of inconsistencies in decreasing significance order. The ability to specify the process model along with the ability to specify the criteria for the assessment of violations of individual consistency rules makes the activities of diagnosing and handling inconsistencies fully tailorable to the needs of specific projects when using Reconciliation. More details on the specification and enactment of the process model of the method may be found in (Spanoudakis and Kim, 2001).

On going work on the framework presented in this paper focuses on its further experimental evaluation. We are also investigating the possibility of expanding it with additional characteristics of model elements as well as with characteristics of entire fragments of models (as opposed to individual model elements) which are related to inconsistencies.

Acknowledgements

The authors would like to thank the British Engineering and Physical Sciences Research Council (EPSRC) for partially funding the work presented in this paper (IMOOSD project, EPSRC grant No. GR/M57422). They also wish to thank Sylvia Mwenya for carrying out some of the experiments whose results have been analysed in this paper, and Dr. Patricio Letelier (University of Valencia) for participating in the experiment described in Section 5.2.

References

Chidamber, S., Kemerer, C., 1994. A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20(6), 476-493.

Emmerich, W., et al., 1999. Managing Standards Compliance. IEEE Transactions on Software Engineering 25(6), pp. 836-851.

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B., 1994. Inconsistency Handling In Multi-Perspective Specifications, IEEE Transactions on Software Engineering, 20(8), 569-578.

Finkelstein, A., Spanoudakis, G.., Till D., 1996. Managing Interference, Joint Proceedings of the Sigsoft '96 Workshops – Viewpoints '96, ACM Press, 172-174.

Gamma E., et al., 1995. Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley.

Hays, W., 1969. Statistics, 3rd Edition, Holt International, SBN 03 910025.

Heitmeyer, C., Labaw, B., Kiskis, D., 1995. Consistency Checking of SCR-Style Requirements Specifications, Proceedings of the 2nd Int. Symposium on Requirements Engineering, IEEE CS Press, 56-63.

Hunter, A., Nuseibeh, B., 1998. Managing Inconsistent Specifications: Reasoning, Analysis and Action, ACM Transactions in Software Engineering and Methodology, 7(4), pp. 335-367

Jacobson I., et al., 1995. Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley.

Kotonya, G., Sommerville I., 1996. "Requirements Engineering with Viewpoints". Software Engineering Journal, vol. 11, n. 1, January, 5-18.

Lamsweerde, A., Darimont, A., Letier, E., 1998. Managing Conflicts in Goal-Driven Requirements Engineering, IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management, November 1998

Lorenz, M., 1993. Object-Oriented Software Development: A Practical Guide, Prentice Hall.

Nuseibeh, B., Easterbrook, S., Russo, A., 2000. Leveraging Inconsistency in Software Development, IEEE Computer, 33(4), 24-29.

OMG, 1999. OMG Unified Modelling Language Specification, V. 1.3a. Available from:ftp://ftp.omg.org/pub/docs/ad/99-06-08.pdf.

Rational, 1998. Rational Rose '98: Extensibility Reference Manual. See also: http://www.rational.com/products/rose/index.jtmpl

Robinson, W., Fickas, S., 1994. Supporting Multiple Perspective Requirements Engineering, Proceedings of the 1st Int. Conference on Requirements Engineering, IEEE CS Press, 206-215

Robinson, W., Pawlowski, S., 1999. "Managing Requirements Inconsistency with Development Goal Monitors", IEEE Transactions on Software Engineering 25(6).

Rosch, E. et al., 1976. Basic Objects in Natural Categories, Academic Press.

Schwanke, W., Kaiser, E., 1988. Living with Inconsistency in Large Systems, Proceedings of the Int. Workshop on Software Version and Configuration Control, 98-118

Shafer, G., 1975 A Mathematical Theory of Evidence, Princeton University Press.

Spanoudakis, G., Finkelstein, A., 1996. Managing Interference, Proceedings of the SIGSOFT '96 Workshops, ACM Publications, 172-174

Spanoudakis, G., Towards an Evidential Significance Diagnosis Framework for Elements of UML Software Models, Technical Report, Technical Report Series, City University, Department of Computing, 1999

Spanoudakis, G., Kasis, K. 2000. An Evidential Framework for Diagnosing the Significance of Inconsistencies in UML Models, Proceedings of the International Conference on Software: Theory and Practice, World Computer Congress 2000, Bejing, China, ISBN 7-5053-6110-4, 152-162

Spanoudakis, G., Zisman, A. 2001. Management of inconsistencies in software engineering: a survey of the state of the art. Handbook of Software Engineering and Knowledge Engineering, Vol. 1, World Scientific Pub. Co, (to appear).

Spanoudakis, G.., Kim H., 2001. Reconciliation of Object Interaction Models, Proceedings of the 7th International Conference on Object Oriented Information Systems (OOIS '01), Calgary Canada, (to appear).

Tegarden, P., Sheetz, S., Monarchi, D., 1995. A Software Complexity Model of Object-Oriented Systems, Decision Support Systems: The International Journal, 13, 241-262.

Appendix: Calculation of Kendall's τ coefficient (for rankings with ties)

The formula used to calculate τ for two rankings r_1 and r_2 with n and k distinct and fully ordered ranking categories that have ties is:

$$\tau = (S_+ - S_-) / ((((N/2)(N-1)) - T_1)(((N/2)(N-1)) - T_2))^{½} \qquad (1)$$

where

- N is the total number of ranked items in each ranking
- S_+ is the number of agreements in the two rankings computed by the formula:

  $S_+ = \Sigma_{i=1\ldots n,\, j=1,\ldots,k}\, c_{ij}\, \Sigma_{u=i+1\ldots n,\, w=j+1,\ldots,k}\, c_{uw}$

  The subscripts indicate the order of categories within a ranking (e.g., the category denoted by the subscript 2 is below the category denoted by the subscript 4 in a ranking).
- S_- is the number of disagreements in the two rankings computed by the formula:

  $S_- = \Sigma_{i=1\ldots n,\, j=1,\ldots,k}\, c_{ij}\, \Sigma_{u=i+1\ldots n,\, w=1,\ldots,j-1}\, c_{uw}$
- $c_{ij}(c_{uw})$ is the number of the items in category i(u) of ranking r_1 and category j(w) of ranking r_2
- $T1 = \Sigma_{i=1\ldots n}\, c_i\,(c_i - 1)/2$, c_i is the number of the items in category i of ranking r_1.
- $T2 = \Sigma_{j=1\ldots k}\, c_j\,(c_j - 1)/2$, c_j is the number of the items in category j of ranking r_2.

The ratio z is calculated according to the following formulas:

$$Z = S/\sigma_s \qquad (2)$$

where

- $S = S_+ - S_-$
- σ_s is an estimate of the standard deviation of S calculated by the formula:

$$\begin{aligned}\sigma_s = \; & ((N(N-1)(2N+5) - \Sigma_{i=1\ldots n}\, c_i\,(c_i-1)(2c_i+5) - \Sigma_{j=1\ldots k}\, c_j\,(c_j-1)(2c_j+5))/18 \\ & + [\,\Sigma_{i=1\ldots n}\, c_i\,(c_i-1)(c_i-2)\,]\,[\,\Sigma_{j=1\ldots k}\, c_j\,(c_j-1)(c_j-2)\,] / (9N(N-1)(N-2)) \\ & + [\,\Sigma_{i=1\ldots n}\, c_i\,(c_i-1)\,]\,[\,\Sigma_{j=1\ldots k}\, c_j\,(c_j-1)\,] / (2N(N-1)))^{1/2}\end{aligned}$$

As an example of calculating the above statistics consider the rankings produced for the 12 inconsistencies of model 11, shown in Table A1. Each inconsistency is denoted by the sequence diagram of the model in which the message that gave rise to it appeared (see column *Sequence Diagram*), the name of the receiver class of the message and the signature of the message (see column *Receiver-class.message-signature*), and the name of the class that sent the message (see column *Sender-Class*). The table also shows the significance scores that the author of the model gave to each of the inconsistencies (see column Sig_a), the rank of the significance category of each inconsistency according to the author's significance scores (see column r_a), the beliefs to the satisfiability of Criterion 6 (see Section 5.2) by each of the inconsistencies (see column *Bel*), and the rank of the significance category of each inconsistency according to these belief measures (see column r_f),

Sequence diagram	**Receiver-class.message-signature**	**Sender-class**	**Siga**	r_a	**Bel**	r_f
AccountTransferSD	AccountQueryBuilder.28.createUpdate(srcAccount,destAccount)	TransactionManager	6	3	.320	6
AccountTransferSD	TransactionQueryBuilder.24.[1st time] createQuery(lastTranNo)	NextTranNo	9	5	.320	5
RequestStatementSD	StatementReqForm.9.displayDefaults (accNoList,fromDate,toDate)	StatementManager	6	3	.231	4
RequestStatementSD	StatementReqForm.9.displayDefaults (accNoList,fromDate,toDate)	StatementManager	6	3	.231	4
RequestStatementSD	AccountManager.6.getAccList (uRefNo)	StatementManager	6	3	.207	3
AccountTransferSD	DatabaseManager.15.executeQuery( )	AccountQueryBuilder	3	1	.182	2
AccountTransferSD	DatabaseManager.25.executeQuery( )	TransactionQueryBuild	3	1	.182	2
AccountTransferSD	DatabaseManager.7.[!exists]create	AccountQueryBuilder	4	2	.182	2
AccountTransferSD	DatabaseManager.8.executeQuery( )	AccountQueryBuilder	3	1	.182	2
RequestStatementSD	TransactionTextBox.20.create (tranText)	Statement	8	4	.154	1
RequestStatementSD	DatabaseManager.14.executeQuery( )	TransactionQueryBuild	3	1	.154	1
RequestStatementSD	DatabaseManager.8.executeQuery( )	AccountQueryBuilder	3	1	.154	1

Table A1. *Author and framework rankings of sample of inconsistencies in model 11*

Based on rankings r_a and $r_{f,}$ the values of Kendall's rank correlation statistics are:

S_+	=	36	T1	=	16	S	=	28	σ_s	=	13.3
S_-	=	8	T2	=	11	τ	=	0.53	z	=	2.1