Classifying Requirement Conflicts for Multi-Stakeholder Distributed Systems

Frank Marschall, Maurice Schoenmakers
Technische Universität München
Lehrstuhl Prof. Dr. Manfred Broy
Software & Systems Engineering
(marschal|schoenma)@in.tum.de

Abstract

Multi stakeholder distributed systems become more and more widespread and raise a lot of integration problems. One problem is that conflicts arise often only at runtime if a single system component is changed. The whole composition of systems then doesn't behave like at least one of the stakeholders expects. The following paper provides a classification of the potential conflicts and gives some guidelines how to handle and overcome these conflicts using this classification. The classification is based on the fact that some parts of a system implementation can be linked to explicit stated stakeholder requirements, while others are just implementation specific parts that are not related to any explicitly stated requirement. Therefore a prerequisite for a successful conflict resolution is the traceability between requirements of a requirements model and the affected parts of an implementation model.

1. Introduction

Resolving requirement conflicts and combining reusable software components are often tasks performed once during the system development. The system is deployed in a controlled environment, and after deployment, each requirement change enforces new requirement engineering and system integration cycles.

Today systems are more and more composed of distributed services which are under control of loosely coupled stakeholders with possibly conflicting interests. The services are deployed independently and combined at runtime. Examples for such scenarios are web service architectures for B2B e-commerce systems with a large set of business partners or enterprise application integration systems which compromise a large set of single applications of different departments.

The result is that the requirements are likely not to be propagated throughout all participating parties, thus conflicts may not appear although they exist. There is no central explicitly coordinated consistent conceptual model of the system at all times. Instead each participant has its own conceptual model. System changes are performed without notice of other stakeholders, which results in unexpected misbehavior.

In this paper we propose a model for the classification of requirement conflicts and show how to reason about conflicts in terms of a conceptual model by using a little example.

2. Example

The following example stems from [7] and describes a web service scenario. There are four stakeholders: the *user* who uses the *UpToTheMinuteNews* news service, *Corporate IT* that provides internet access for the user through a proxy, and the *AWS company* that provides a caching proxy that is used by Corporate IT.

From the moment when Corporate IT starts using the AWS proxy, the user experiences that the news form UpToTheMinuteNews isn't up to date any more. Obviously this does not meet the user's requirements while Corporate IT and AWS might not even have reasoned about this topic.

3. Requirements

In our model we suppose that stakeholders have requirements which are basically statements about the systems in their scope. These requirements form a requirements model of the desired system. This model is refined into an implementation model that is enriched by design decisions which are not considered in the conceptual system model. When the models are formal an approach like considered by the Model Driven Architecture (MDA) approach [5] may be chosen to derive platform specific models from more abstract platform independent models and computational independent models. Ideally the requirements can be

traced so that one knows from which parts of the requirements model a certain part of the implementation model stems. Since the implementation model is a refinement of the requirements model it has in general properties that were never explicitly required and thus fulfil never stated "requirements".

Thus the implementation model can be partitioned into two parts: one contains the model elements that can be seen as a direct refinement of the conceptual model. The other part contains the elements that were not explicitly specified in the conceptual model. This is typical since requirement models are usually underspecified, i.e. they often leave (intentionally or not) room for design decisions.

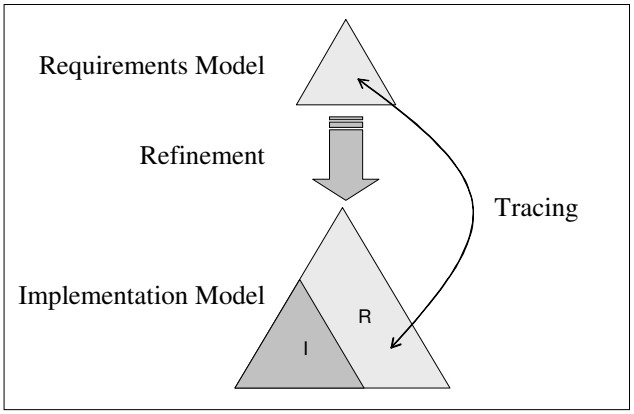

Figure 1. A single system. Assumption: the implementation satisfies the requirements

Figure 1 depicts these facts. The implementation model comprises a part R that was derived from the requirements model and a part I that cannot be mapped to any elements of the requirements model but is necessary for the system to work.

When combining two systems, there exist overlapping parts that both systems have to deal with in the requirements and in the implementation model. For example the requirements models of both systems have to consider the common goals of their collaboration, exchanged data types, messaging mechanisms etc. Often the latter arise only in the implementation model because they didn't seem to be relevant to the stakeholders and thus the decision was left to the developers.

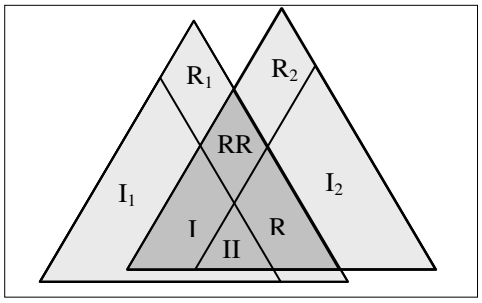

Figure 2: Conflicts between two systems can occur in implementation areas RR, II, IR/RI.

Figure 2 depicts the conflicts that may arise if two participants integrate their systems: the following conflicts can occur simultaneously. There may occur a conflict between parts of the systems that

- reflect the requirements of both participants: **RR conflicts**
- where not considered by the requirements of any participant**: II conflicts**
- where only considered by the requirements of just one of the participants: **IR/RI conflicts**

Requirement conflicts (RR) are fundamental conflicts which must be resolved before the two parties can form a system satisfying to both parties. Normally one tries to prevent these conflicts by exchanging some kind of requirements model description concerning the overlapping parts. For example IT and AWS sign a contract in advance, which contains the rules of operation between these parties. Identifying such a conflict at runtime must result in an adaptation of the requirements at one or both sides or more likely results in dissolving the contract between the two parties.

Conflicts concerning parts of the implementations that are not considered in the requirements model of at least one stake holder, the II and RI/IR conflicts, can be overcome by adapting the implementation without changing the original requirements. However such a conflict indicates that the requirements specification is incomplete and should be completed. Therefore mechanisms are needed that identify the context of the conflicting parts of the implementation model and identify the appropriate area of the requirements part where information is missing. For example if the requirements model doesn't state anything about the message exchange between two components and during operation it turns out that this message exchange fails such a mechanism could lead a stakeholder to the involved components that failed to communicate. Thus the stakeholder would be confronted with the problem in terms of the requirements model, not with a technical error. He can make a decision at the high level requirements model and refer it to the developers. These

refine the new requirement into a consistent solution, presumed that the requirements model is not still underspecified for the collaboration of the components.

In case of a legal contract one party may enforce the other party to adapt to an implementation. However, this is not always possible or feasible because each partner controls which implementation is deployed, and the implementation can be chosen at the other party because of other internal requirements. Conflicts in these classes occur often by not having explicitly modelled the requirements on one or both sides.

Some RI/IR conflicts between requirements of one party and implementations of the other party can be very problematic and should generally be avoided. One party has a requirement accidentally fulfilled by the implementation of the other party while it was not explicitly guaranteed by the conceptual model or contract. The other party can change the implementation at any time which may cause the requirement to become unfulfilled. Therefore such requirements should be explicitly stated and exchanged in advance to lift the fulfilled feature up to the RR class.

In the previously described example, where the user experiences that the news form UpToTheMinuteNews is not up to date any more, because Corporate IT started to use the AWS proxy, there are seven possible reasons given in [7] for the conflict. We now look at these reasons from the classification perspective:

1. AWS's contract has fine print explaining that it does not guarantee freshness of its pages. In this case the requirements model of the user / Corporate IT is underspecified, thus this is an IR conflict. AWS has specified a requirement the user / Corporate IT didn't consider. However adding the new requirement that the information provided by the AWS proxy must be fresh enough would yield to a RR conflict in the requirements specifications that must be eliminated by the involved stakeholders.

2. AWS's contract is purposely inaccurate or unclear at this point. Here two cases can occur: The AWS implementation is caching and delaying requests by purpose then there is an AWS internal requirement hidden for the IT department, which is not stated in the external contract. In this case there is an IR conflict as the IT department did not specify its requirements sufficiently. If the AWS implementation causes the conflict just because the developers choose the implementation accidental, then both parties did not specify the requirements thus an II conflict occurred.

3. AWS's implementation is buggy, old, or incorrectly configured so that it does not honour "no-cache" header information. In this case the refinement of the AWS requirements model to its implementation model failed. Such conflicts must be avoided by efficient testing of the implementation model against its requirements model. The other party can in this case insist on the adaptation of the implementation if possible.

4. U2M's implementation is buggy or incorrectly configured, so that it does not produce "no-cache" headers with its pages. This is the same case as (3).

5. IT failed to read or correctly interpret the AWS contract. In this case the conflict detection between requirements models failed. The RR conflict was not detected. With more formal requirements specifications (e.g. B2B Specifications like ebXML [3]) some of these conflicts can be avoided. However there may still be cases when these specification languages are not expressive enough to formalize all desired requirements.

6. IT failed to realize that some of its users required freshness. In this case the requirements model of IT or that of the users is underspecified (if it doesn't state anything about the freshness of information) or it is simply wrong and needs to be reworked.

7. The user failed to communicate to IT that it needed access to a time critical web site, even though IT surveyed its users on this point at some time previously. Again the requirements model of IT is wrong and needs to be reworked. Both IT and the user didn't specify parts of their overlapping requirement model in an explicit contract.

4. Position

In our opinion the following criteria must be fulfilled to correctly handle occurring conflicts:

- *A) Differentiate between required (R) and not required (I) implementation parts at both parties.* One must be able to differentiate between the system properties which originate from real requirements and those properties stemming from implementation refinement steps chosen by developers for technical reasons unrelated to any stated requirements.

- *B) Tracing back from implementation components to the requirement model.* Because conflicts arise at the implementation level tracing a requirement from the

implementation level up to the original requirement level becomes an essential feature. The affected implementation parts of the conflict at implementation level must be related to the requirements in the original requirement model. This allows an expression of the conflict at the language level used by the stakeholders. This in turn allows an effective requirement conflict resolving discussion.

We propose that each partner formulates requirements explicitly and forms a conceptual model to relate elements of the conceptual model to system components as well as to ensure traceability.

Traceability is very important for open distributed multi stakeholder systems. Exchanging model information may be used to prevent RR conflicts but cannot prevent II, RI/RI conflicts. Furthermore as requirements are often not made explicit and requirements and implementations change over time *requirement conflicts may arise only at runtime in form of implementation conflicts*. Only traceability between conceptual model elements and system components of the implementation can then be used to identify to which classes RR, RI, IR, or II the conflict really belongs. To which class a conflict belongs in turn shows how to resolve it and this may reveal the legal and financial consequences.

A conflict is classified as follows: If the conflicting system component is related to a requirement at each party, then the conflict may belong in the RR class, if these requirements are conflicting. In this case there is a fundamental problem. If the conflicting system component is related to a requirement at just one party, the conflict belongs to the RI or IR class and without any related requirements at either side it belongs to the II class. In any case a conflict belonging to the RI/IR or II may also point to an incomplete conceptual model. The parties then can at least pinpoint the problematic areas and discuss resolution possibilities.

In the research project KOGITO [6] the authors address some of the topics mentioned here. The scope of KOGITO is requirement engineering for multi stakeholder B2B internet systems. Typical problems are the integration of different existing open distributed systems. To ensure the above mentioned tracing capabilities KOGITO has defined a multilevel conceptual model. Requirements in documents individually reference subsets of the conceptual model elements. The levels of the conceptual model range from coarse grain business process areas to fine grained message exchange. During system development the refinement steps result in linking the different levels of the conceptual model. The steps become traceable and fine grained system components become related to single requirements at the conceptual level. This ensures the traceability and the capability to differentiate between those implementation parts directly linked to requirements and those not linked to requirements. Furthermore the integration of formalized ebXML [3] business process descriptions at a middle level allow to check and ensure consistent overlapping requirements and helps avoiding RR conflicts by reusing predefined business processes as some kind of contracts.

Until now we discussed cases where both parties were able to get in contact to each other directly, can sign contracts and resolve conflicts in a direct discussion with stakeholders.

This is only possible for open distributed systems with a limited number of participants and a rather stable requirement model, where stakeholders may have a contractual relationship. For fine grained fast changing systems with a high number of participants tracing conflicts back to requirements and resolving conflicts in discussions is not feasible. Especially as the stakeholders may not have any direct contractual relationship. Examples for such systems are large distributed web service [2] or Jini [9] based systems. To trace back conflicts or to generate conflict resolutions automatically would require a high degree of formalization [8], [4]. For such systems it is probably more feasible to avoid conflicts and to check for RR conflicts automatically up to a certain level instead of resolving them. Thus stakeholders will have to agree on standards or models for which they claim to provide a correct implementation.

While the proposed principles and conflict classes do not change, the way conflicts and requirements and are identified and are resolved would change: As RR conflict checks must be performed frequently and automatically. We propose that implementations are accompanied by an explicit formulated abstract conceptual model which expresses the requirements and defines the collaboration between the participating systems (for example as an ebXML business process description [1] with an appropriate role profile). Each implementation would thus be accompanied by a simple formalized version of a requirements model of Figure 1. Before components would be combined the models then could be checked automatically for RR conflicts, this roughly would reflect the case of signing a contract. If no conflicts arise the implementations could be combined. Conflicts of any of the classes II, RI, RI and RR could still occur. However the change of conflicts in the RI and IR classes would be lowered as the requirements would be quite explicitly defined and modeled and RR conflicts could only stem from requirements which cannot be formulated in the formalized requirement description model.

The problem of inconsistencies in the conceptual model respectively between the contract and the implementation is not addressed by this proposal. Such

problems occur because the implementation doesn't fulfill the requirements either by purpose (deception by a stakeholder) or by failure would still arise. While it could be guaranteed that such problems would not occur by proving each refinement step in a formal way, it would probably more feasible to combine the requirement models and their implementations with certificates or ratings. A trust relation could be established in the following way: A third party checks if the requirements model of a stakeholder is fulfilled by the stakeholders implementation. If both, the stakeholder using the system and the stakeholder providing the system, trust this third party, then they can be confident to a certain degree that conflicts will not arise because of deception or failure.

5. Conclusion

The proposed conflict classification allows to reason about occurring conflicts, for example if a conflict is a simple technical implementation issue or a fundamental requirement conflict. It therefore provides valuable hints how to solve these conflicts and to consider the conflict implications.

To classify conflicts the basic capability identified was traceability of refinement during development, how requirements are related to system components of the implementation. This ensures the capability to trace a conflict back from the implementation to the requirement level. This in turn is crucial to identify if a fundamental requirement conflict occurred or merely a technical failure.

The best way to handle conflicts is to avoid them upfront by formulating contracts on a conceptual requirement level. We gave an outlook how more or less formalized conceptual models like ebXML process descriptions could help avoiding requirement conflicts

References

[1] ebXML Business Process Specification Schema Version 1.01 6, http://www.ebxml.org/specs/ebBPSS.pdf, 11 May 2001

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2):86{93, March 2002

[3] Electronic Business using eXtensible Markup Language, UN/CEFACT and OASIS, http://www.ebxml.org

[4] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency Handling in Multi-perspective Specifications", IEEE TSE, 20(8): 569-578, 1994

[5] MDA Guide Version 1.0, Joaquin Miller and Jishnu Mukerji, http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf, Mai 2003 OMG

[6] Towards Model-Based Requirements Engineering for Web-Enabled B2B Applications, Frank Marschall, Maurice Schoenmakers, Proceedings 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), p. 312, April 07 - 10, 2003

[7] Workshop on Requirements Engineering and Open Systems (REOS), Home Page, http://www.cs.uoregon.edu/~fickas/ REOS/, 2003

[8] Robinson, W.N., Volkov, S., Conflict-Oriented Requirements Restructuring, GSU CIS Working Paper 99-5, Georgia State University, Atlanta, GA, April 9, 1999

[9] Jim Waldo. The Jini Architecture for Network-centric Computing. Communications of the ACM, pages 76--82, July 1999