Reverse Engineering Tool Requirements for Real Time Embedded Systems

Brice Govin [1,2]
Arnaud Monegier du Sorbier [1]

[1]THALES AIR SYSTEMS,
Parc tertiaire SILIC,
3 Avenue Charles Lindberg 94628 Rungis Cedex

Nicolas Anquetil[2], Anne Etien[2],
Stéphane Ducasse[2]

[2]RMod team, Inria Lille Nord Europe,
University Lille 1, CRIStAL, UMR 9189,
59650 Villeneuve d'Ascq, France

Abstract

For more than three decades, reverse engineering has been a major issue in industry wanting to capitalise on legacy systems. Lots of companies have developed reverse engineering tools in order to help developers in their work. However, those tools have been focusing on traditional information systems. Working on a time critical embedded system we found that the solutions available focus either on software behaviour structuring or on data extraction from the system. None of them seem to be clearly using both approaches in a complementary way. In this paper, based on our industrial experiment, we list the requirements that such a tool should fulfil. We also present a short overview of existing reverse engineering tools and their features.

1 Introduction

During their lifetime, systems face constant evolution, for example to fulfil new requirements. To keep useful they have to be maintained. However, maintenance requires a deep understanding of the systems that can be obtained for example using reverse engineering techniques. Some systems are data oriented and are mostly developed using object oriented paradigm. Other systems are behaviour oriented and are typically programmed using procedural paradigm. In this paper, we will focus on this second type of system and more precisely on real-time critical embedded ones.

These systems have specific constraints and characteristics. They are behaviour oriented in order to answer to time requirements, real time and hardware constraints. Data on the contrary were often not structured following domain abstractions, but rather to attend the needs of one functionality or the other. A domain abstraction of the data is nevertheless required to understand and document fully the system.

We tried traditional reverse engineering tools available on the market or in academia on several industrial real time critical embedded systems. However, they are mostly based on structure recovery in the object-oriented paradigm and do not fit in our cases. This paper aims to highlight features that a tool should provide to reverse engineer real time embedded systems.

The next section (§2) presents the features required by a reverse engineering tool dedicated to real time critical embedded systems. Section §3 reviews several existing reverse engineering tool in the light of these needed features. Section §4 draws some conclusions and presents future work.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

2 Features for a reverse engineering tool

From a review of literature and based on our experience reverse engineering critical systems with existing tools we identify some features that a reverse engineering tool should provide to fit behavioural systems specificities.

Ducasse and Pollet provide a state of the art on software architecture reconstruction [DP09]. They propose to classify approaches according to five criteria: goal, process, input, technique, and output. Based on these criteria, we identify the following features.

Process: Ducasse and Pollet state that "because hybrid processes reconcile the conceptual and concrete architectures, they are frequently used to stop architectural erosion". This process seems to fit better the case of reverse engineering a live system to keep it evolving and a tool should support that.

Output: It is important to keep in mind that despite the goal of extracting an abstract understanding of the system, the source code remains the main manipulated artefact. Thus, reverse engineering tools must provide *source code visualisation* like in a text editor.

According to [DP09], a lot of existing approaches provide graphical representation of system views. This is very common in software engineering at large. Thus, some *graphical visualisation* capability seems essential for any reverse engineering tool.

Input: The physical organisation of the system (that we call *concrete code structure*) corresponds to the organisation on the disk in term of files and folders as well as grammatical structure from the programming language. It often reveals structural information and a tool should be able to represent this information, map it to the source code and manipulate it.

In parallel with this, we consider a *concrete data structure.* It corresponds to the data structure extraction defined in [HHH+00]. Data is a very important aspect of any software system. Real time, embedded, systems will typically not offer data structures easily mappable to domain concepts because the data are typically implemented so as to facilitate the implementation of the behaviour.

Human expertise specifying a conceptual architecture is very helpful when it is available [DP09]. This logical structuring of the system can be completely different from the concrete code structuring. We call this the *abstract code structure* and again should be represented, mapped to the system and manipulable. Due to the type of system we target, this structure is typically behavioural oriented.

Similarly, we consider an *abstract data structure.* It is similar to the data structure conceptualisation specified in [HHH+00], and aims to define data structure in abstract terms, typically closer to domain concepts.

Dependency analysis aims to analyse dependencies either between data (*e.g.* sub-typing, reference) or between behavioural entities (*e.g.* invocation).

The five previous features focus on a single type of artefact either data or behaviour. However, data are manipulated by the behaviour *e.g.* through arguments in method invocations and as stated earlier, it is important to be able to consider data and behaviour in a complementary way. *Data/behaviour relationships provider* aims to highlight these relationships.

Techniques: Manipulating code either directly or through abstraction requires to be able to retrieve a given element by reference or by querying the representation. Therefore a *search* feature is needed.

Output: In the conformance output, [DP09] classify the ability to specify conformance rules and to validate them. These rules can be relative to either data, behaviour or both. A reverse engineering tool should therefore have a *rule creator and checker.*

Goal: In order to target the co-evolution goal, implementation and abstract representation that evolve at different speeds should be synchronised [DP09]. A tool should offer the possibility to *implementation an abstract structure* by modifying directly the source code according to changes made at higher abstraction level.

Finally it seems desirable that the tool be *open to user extension* and give its users the ability to personalise the tool for example by creating their own queries or visualisation and not only used the provided ones. Another example would be to allow the users to connect the reverse engineering tool to any other that they already use.

3 Reverse Engineering Tools

In this section, we look for the features previously defined on several tools available on the industrial market or in academy: Understand [Sci], Agility [Agi] , CodeCase [Cod] are industrial tools, Rigi [SWM97] is an academic tool and Moose [DLT00] is a meta-tool. Each of them provides code extractor for non object-oriented languages. However, the industrial tools focus on data and are relatively limited concerning behavioural and so also concerning data/behaviour relationship. The two academic tools are more generic, they can be adapted to focus on behaviour. Table 1 lists the features we found in each tool.

Table 1: Comparison of a few reverse engineering tools. (Agil=Agility, CC=CodeCase, Und=Understand) ("v" means that the tool provides the feature, "∼" that the tool can be adapted to provide the feature and a blank cell that the feature is not provided. "BU" stands for "bottom up".)

Features	Agil.	CC	Und.	Moose	Rigi
Process choice	BU	BU	BU	BU	BU
Source code visualisation	v	v	v	∼	∼
Graphical visualisation	v	v	v	v	v
Concrete code structure	v	v	v	v	v
Concrete data structure	∼	∼	∼		
Abstract code structure					
Abstract data structure					
Dependency analysis	v	v	v	v	v
Data/beh. relationships provider	v	v	v	v	v
Search	v	v	v	v	v
Rule creator and checker					
Implementation of abstract structure					
Open to user extension				v	v

4 Conclusion and future work

Based on literature review and our experience in reverse engineering real, industrial, time critical, embedded systems, we identified required features for a reverse engineering tool dedicated to this type of system. The defined features concern a system's behaviour, data and the relationship between them. They also correspond to different abstract representation of the code and the way to either view it or query it. Each of these features should be mapped directly or indirectly to the code, since in the end, it remains the main artefact.

Existing industrial and academic tools are then evaluated according to these features. Even if each of these tools provide code extractor for non object oriented languages, none of them provides all features. Our future work is to develop a more complete reverse engineering tool dedicated to real time systems. For this purpose, we will base our work on existing techniques deployed in software architecture reconstruction or in data reverse engineering.

References

[Agi] Obeo Agility. `http://www.obeo.fr/fr/produits/obeo-agility`.

[Cod] CodeCase. `http://codecasesoftware.com/offers-expertise/transformation.html`.

[DLT00] Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, volume 4, 2000.

[DP09] Stéphane Ducasse and Damien Pollet. Software architecture reconstruction: A process-oriented taxonomy. *Software Engineering, IEEE Transactions on*, 35(4):573–591, 2009.

[HHH+00] Jean-Luc Hainaut, Jean Henrard, Jean-Marc Hick, Didier Roland, and Vincent Englebert. The nature of data reverse engineering. In *Proc. of Data Reverse Engineering Workshop (DRE)*, pages 1–10, 2000.

[Sci] SciTools. `https://scitools.com/`.

[SWM97] Margaret-Anne D Storey, Kenny Wong, and Hausi A Müller. Rigi: a visualization environment for reverse engineering. In *Proceedings of the 19th international conference on Software engineering*, pages 606–607. ACM, 1997.