Reverse Engineering with Petri Nets

Walter Keller
PhD student at the Department of Information Technology, University of Zurich, Switzerland
Steinackerweg 33, CH-8304 Wallisellen, email: wkeller@datacomm.ch

Abstract

With the emergence of Petri nets in practical applications the need to reverse-engineer them arises. Folding based reverse-engineering techniques are crucial for Petri nets. But after a translation step they offer novel analysis capabilities for other systems. Such a translation makes Petri nets a powerful and intuitive engineering metaphor outside their traditional strength for concurrency.

We present a folding-based algorithm which transforms an unstructured flat net into a coloured net. In reverse engineering terms, it recovers a high-level design, a structured specification and a data model from an existing system. Both the algorithm and the translation to Petri nets allow many variations for adaptation to different tasks. Moreover, the cost is almost linear, thus ensuring scalability.

***Keywords**: Petri nets; coloured nets; folding; reverse engineering; design recovery; engineering metaphor;*

1 Introduction

The effort to understand existing software systems is well known to be a key contributor to software costs. Experience has shown that every new programming technique and design method needs specific reverse engineering methods. The use of Petri nets was emerging during the last years and this implies the importance of reverse engineering methods for Petri nets. This is the subject of this work.

A prerequisite for reverse engineering is the selection of the kind of model to recover. We differentiate two structuring principles: folding and clustering.

In clustering, directly connected entities are grouped together. Neighbourhoods are collapsed into a single object which also swallows relationships. Foldings, on the other hand, merge similar objects and, separately, similar relationships. A model object stands for similar objects and hence inherits many properties.

Clustering respects vicinity, is the usual method of breaking complex systems down into subsystems and is predominant in reverse engineering. But a clustered node is neither a transition nor a place and thus incompatible with the rigorous semantics of Petri nets. Therefore for Petri nets folding is preferred which finds a very convenient representation in a coloured Petri net. Nevertheless, clustering is an indispensable supplement for practical applications.

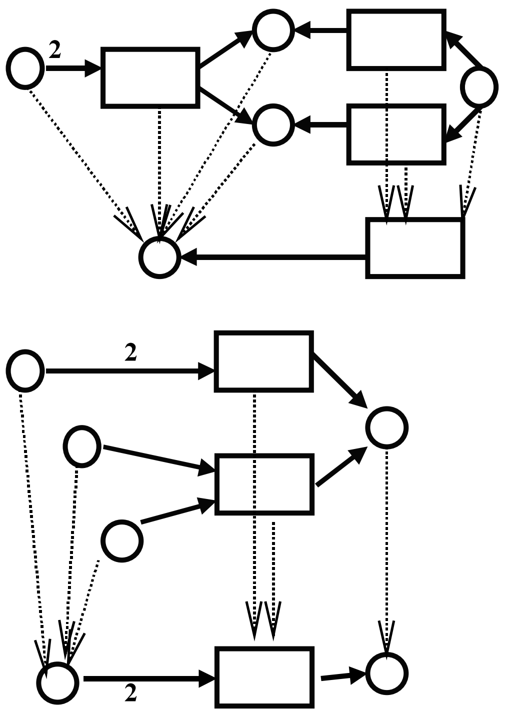

Figure 1. Clustering (top) and folding (bottom).

This work concentrates on folding-based Petri-net techniques for reverse engineering - clustering-based ones share many similarities with well-known methods. This paper proposes to translate a legacy system to a Petri net and presents an efficient and flexible algorithm to deduce a coloured net from it. This yields a novel and powerful reverse engineering method. It recovers high-level design and specification information such as the data model.

2 Modelling with Petri Nets

Figure 2 shows a small fragment of a spare-part ordering system as a Petri net N^u. The author was involved in implementing it in a 4GL procedural language for a loom manufacturer. A Petri net consists of transitions painted as boxes and places painted as circles which may contain tokens. A transition may occur which removes tokens from its pre-places and puts tokens on its post-places. Formally a net is a pair of functions pre and post which map a transition to its pre and post multiset of places ([6]).

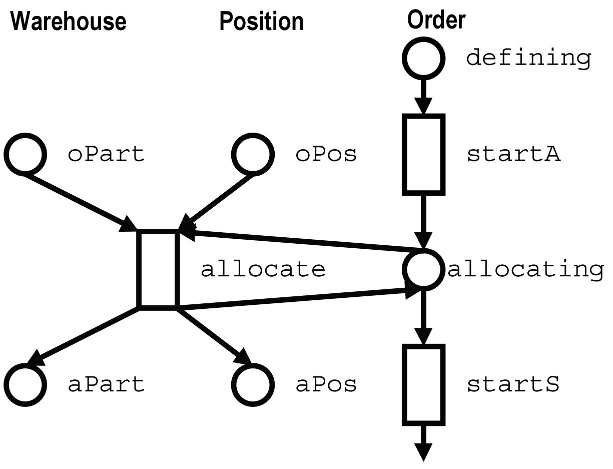

Figure 2. The design net N^u of the spare-part system.

An order consists of positions, each containing a spare part. After the order has been defined parts get allocated: An occurrence of the transition `allocate` moves a token from `oPart` to `aPart` indicating the reservation of this part in the warehouse. It also moves a token from `oPos` to `aPos` showing the reservation of the corresponding position. The occurrence of the transition `startS` initiates the next phase of order processing.

A colouring of this net reflects that an order may contain several positions and the system consists of many orders. It is described by a colouring morphism c: $N^f \to N^u$ from a flat net N^f. Informally c maps places to places, transitions to transitions and preserves pre and post (e.g. c $pre^f = pre^u$ c). Hence the origins (by c) of `oPart` and `aPart` respectively correspond to the set of spare parts. Transitions are assigned colours which allow valid state changes, e.g. `ship` may move the part belonging to a specific position of the corresponding order. N^u is a high-level design and the morphism c a specification which contains information such as the data model.

If this coloured net is the design, an implementation may translate places to database tables and transitions to procedures which update the database as described by the pre and post-functions. This is straightforward. Hence, coloured nets qualify as design formalism outside the area of Petri nets. They show the interplay of structural, functional and dynamic aspects in a compact and graphically attractive way.

Reverse - a legacy system may be translated to a net. This could be done by the above method, but, we get a more realistic reverse engineering task by translating:

- DB transactions to transitions,
- disk addresses (tuple identifiers) to places and
- the update of tuples by transactions to pre and post.

Again, the translation is straightforward. Reverse-engineering the colouring of such a net deduces design and specification from low-level runtime information.

But we want to stress that we have described just two of many modelling methods and for each of them colourings express semantics relevant for certain applications. Hence reverse engineering a colouring is valuable for many tasks.

3 Unique Reduction

The metaphor for the reduction algorithm is another use of the word folding. Just like folding a shirt, two adjacent parts which fit together are overlaid. Figure 3 shows how neighboured transitions and their surrounding places are overlaid. The algorithm works iteratively: similar subnets are overlaid when they become adjacent, and it stops when there are no more such adjacent similar subnets. Here subnets consist of a single transition and its adjacent places and they are similar if they are isomorphic.

The reduction algorithm computes the net shown in Figure 4 from unfolded flat nets of different sizes of the spare-part order system from Figure 2. It looks quite similar to the design and gives a clear picture which offers a good starting point for further analysis. The algorithm cannot distinguish parts and positions because swapping them is an automorphism of the subnets of `allocate`.

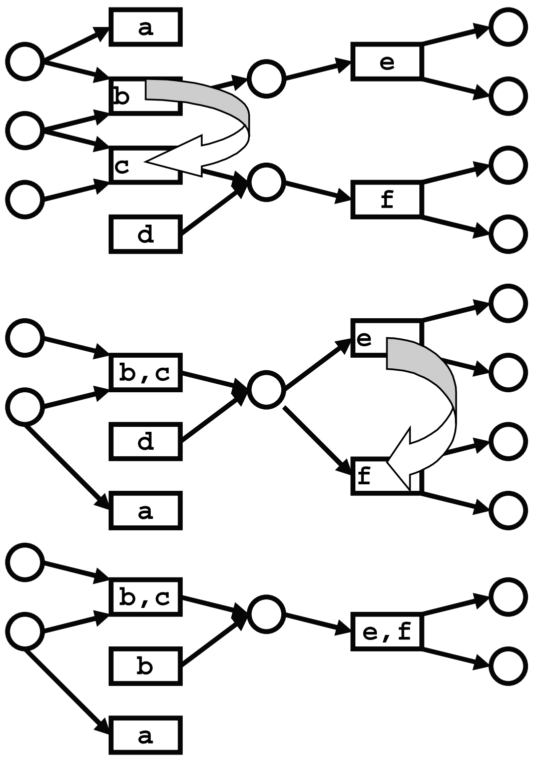

Figure 3. Reduction by folding.

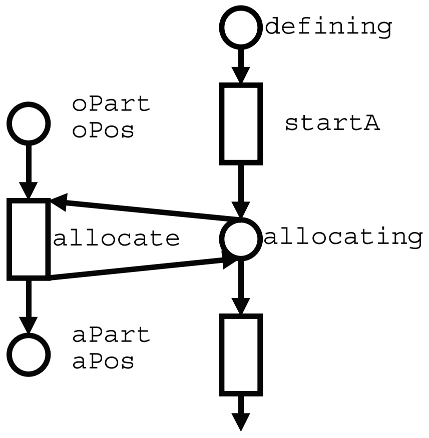

Figure 4. Result of the simple reduction.

On this basic level the algorithm may be modified in three dimensions:

- Similarity: which subnets are merger candidates? Should subnet isomorphism be weakened or strengthened by additional requirements?
- Adjacency: how near should two subnets be? Or should adjacency simply be ignored?
- Choice: which mergers allowed by the above two criteria are selected?

Internally, the algorithm computes a sequence of intermediate reductions. In such a reduction, it checks the arcs around a node. If the origins of these arcs are connected compatibly to similar transitions they and corresponding attached places become merged. This yields the next reduction wherein the arcs around the newly merged places are recursively checked. For reasons of performance, it is essential to minimise repetitions of mergers of an arc caused by successive mergers of its place. Also, the intermediate reductions are stored in an incremental way that, nevertheless, allows fast read access. For this equivalence relations are used.

With such optimisations and with the assumption that the cardinality of the pre- and post-multisets of transitions is limited by a constant, the cost of the algorithm is

$O(e \log(\min(\text{maxDeg}', |P|))\, \gamma)$

with e the number of edges in the source net, maxDeg' the maximum number of arcs of a place in the reduced net, |P| the number of places of the source net and γ a slowly increasing inverse of the Ackermann function, which does not surpass 3 in any real case.

The details of the algorithm and the proof of the cost limit may be found in the author's PhD thesis [4]. It also shows that in terms of category theory the algorithm is an iteration of universal constructions, so-called coequalisers, and is itself universal. This, for instance, yields uniqueness and eases optimisation.

4 Relationship Analysis

The integration of relationship analysis in the reduction algorithm allows to separate parts and positions and, furthermore, to recover the colourings.

For the first task we use the fact that although parts and positions may be swapped by an automorphism of `allocate` they are in a 1:n relationship. A tricky method allows to compute this asymmetry beforehand and to feed it as the similarity criteria to the algorithm. Because realistic 1:n relationships contain sporadic 1:1 pairs arbitrary decisions are needed to avoid that the overall asymmetry is compromised. The enhanced algorithm reverse-engineers our example system to the same diagram as in Figure 2.

Relationship analysis is used again to recover colouring information by the following steps:

- Compute the relationships from the reduction, i.e. group the start and end nodes of all arcs in the source that coincide in the reduction.
- Optionally compose relationships to longer paths. Not limiting the path length leads to combinatorial explosion.
- Classify the relationship cardinalities.
- Select the colour sets using relationships. Reduced nodes in a 1:1 relationship use the same colour-set. This method relies on stable properties of the analysed net - not on an accidental equality of cardinalities.
- Colour the arcs using a classification of relationships between colour-sets. The 1:1 relationships from the previous step yield identities. The algorithm classifies relationships in terms of equality, inclusion and composition.

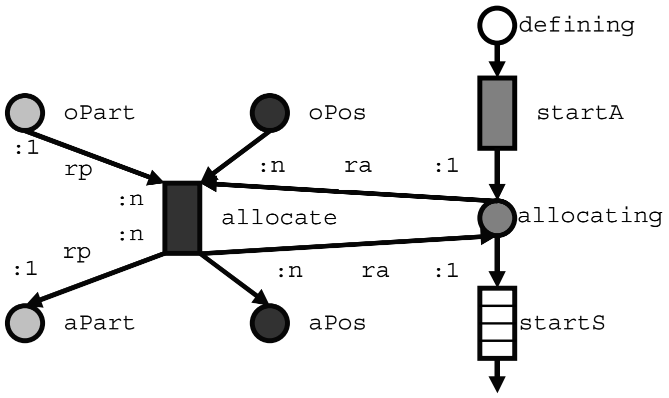

Figure 5. The analysis of the enhanced algorithm.

Figure 5 shows the fruits of these enhancements on the spare-part system. A colour set is painted as a pattern inside the node symbol. An arc is inscribed with the name and cardinalities of the colour relationship:

- `rp` shows which part belongs to a position
- `ra` shows which order belongs to a position

In fact, this diagram is better than the design in Figure 2. The colour-sets are now precise and the colour relationships are explicitly classified - before this was left to the application know-how of the analyst. This once more shows that reverse engineering requires a more precise understanding than normally goes into a design. Also it indicates that our Petri net notation is potentially more precise than the usual design diagrams. It is a strength of the algorithm presented here to recover such hidden information.

5 Applicability

Lack of resources prevented the application of the presented reverse-engineering method to real-world systems. We had to restrict to small-scale experiments with a prototypical implementation.

Most experiments were done on a system containing the fragment from Figure 5. Unfolded nets of up to 40000 arcs (the memory limit of the prototype) were successfully analysed yielding a reduced net of 27 arcs. If the complexity of the unfolding was increased the analysis contained additional nodes. For the running example these additional nodes have application significance. Also, we showed how to reduce them by additional heuristics.

Furthermore, we analysed an embedded system built with a clustering-based component structure which is completely different from our structuring principles. Still, the algorithm detected non-trivial components.

Remember that the algorithm has three dimensions of variations and a preceding translation step which also is adaptable. From this and the mentioned experiments we draw the hypothesis that the unchanged algorithm offers a good starting point for many reverse-engineering tasks and that it can be integrated with domain-specific heuristics to produce deeper insights. The latter requires careful balancing of different factors, as was experienced with the relationship-analysis technique.

6 Previous Work

Although abundant literature is available about Petri nets and reverse engineering, the intersection between the two fields is surprisingly small.

In software engineering, Petri nets are sometimes used to model dynamic aspects or the software-engineering process. They appear fairly often in business re-engineering, but none of the tools we are aware of includes Petri-net based analyses for reverse engineering. A few authors model functional or structural aspects with nets, e.g. [3] and [1]. In [2] universal constructions in algebraic Petri net categories are used for software engineering.

7 Conclusion

We presented a folding-based Petri-net algorithm which can recover a high-level design and a structured specification from an existing system. It revealed the functional and structural architecture as well as hidden information not explicitly represented in the design. Although this excellent result has only been verified at a few small cases we hope that it may prove useful for real-world problems. This is backed by three dimensions of variations which the algorithm allows and a preceding translation step that turned out to be intuitive and flexible.

The high speed of the algorithm - almost linear in the input size - represents an attractive feature for reverse engineering, which is notorious for having to combat combinatorial explosions. The algorithm is therefore scalable and may also be used for lightweight analyses.

This work reports some results of the author's PhD thesis [4]. Among other things it defines Petri net categories that connect clustering and folding as well as structure and behaviour (also in [5]). There are many areas for further research, most obviously, our methods should be validated and enhanced by tasks from different domains. Conversely, a rather speculative idea: we used folding-based Petri-net techniques for reverse engineering – there must be other such unexpected applications.

The dialectic between static graph and behaviour is one of the essentials of net theory. Although, obvious correspondences exist in software engineering this work could not make use of them. Translating a system to a bipartite graph offers already powerful modelling and analysis features. In terms of intuition, however, Petri net dynamics behind the static structure is the driving force. This research convinced us that folding-based Petri net methods are useful in reverse engineering and it is worthwhile to dedicate further research to them.

8 References

[1] Yi Deng, S. K. Chang, Jorge C.A. de Figueired, Angelo Perkusich. Integrating Software Engineering Methods and Petri Nets for the Specification and Prototyping of Complex Information Systems. 206 – 223 in Application and Theory of Petri Nets 1993. Springer, LNCS.

[2] Hartmut Ehrig, Maike Gajewsky, Sabine Lembke, Julia Padberg, Volker Gruhn. Reverse Petri Net Technology Transfer: On the Boundary of Theory and Applications. TR 97-21. TU Berlin. 1997.

[3] Olaf Fricke. Data Encapsulation and Data Abstraction with Petri Nets - a Graphical Visualization of Modules. PNSE'97. Uni Hamburg, Informatik, Bericht Nr. 205. 1997.

[4] Walter Keller. Petri Nets for Reverse Engineering. PhD Thesis. 1999. For submisson to IfI, University Zürich. 1999.

[5] Walter Keller. Clustering for Petri Nets. 1999. Submitted to Theoretical Computer Science. Elsevier.

[6] Wolfgang Reisig, Grzegorz Rozenberg (eds.). Lectures on Petri Nets I: Basic Models. 1998. Springer, LNCS 1491.