

Informal Introduction to the Feature Structure Nets Tool

A Tool for Process and Information Modeling

Frank Wienberg², Michael Duvigneau¹, and Heiko Rölke¹

¹ University of Hamburg, Department of Informatics
{duvigneau,roelke}@informatik.uni-hamburg.de

² CoreMedia AG
frank.wienberg@coremedia.com

Abstract. This paper presents a modeling technique that combines process-oriented modeling with information-oriented modeling. The modeling technique is well suited for modeling distributed systems, especially information systems. It is backed by a Petri net formalism (called FS Nets) that uses feature structures as tokens as well as to describe transition rules. Along with the formal definition we present a graphical notation that is supported by the tool RENEW [1].

Keywords: High-level Petri nets, feature structures, basic FS Nets, information-oriented modeling, workflow modeling, Renew, UML

1 Introduction

The idea of the FS Nets (Feature Structure Nets) modeling technique presented in this paper is to combine process-oriented modeling (Petri nets) with information-oriented modeling (feature structures). Petri nets are a well-established modeling technique for processes of concurrent or distributed systems. They are theoretically founded, and come with a graphical notation that is simple to use. Feature structures also have a firm theoretical foundation. We build our technique upon the logic of typed feature structures defined by Carpenter in [2]. With typed feature structures, we can express many aspects that are covered by UML class diagrams and object diagrams, such as types, attributes, generalization and specialization, associations. So we can naturally use a UML-like notation as graphical notation for feature structures.

The most important operation on feature structures is unification. In the FS Net formalism, unification is used to merge the information of multiple input tokens and to extract parts of the information for output tokens. The unification can also be used to join information of multiple copies of one structure that have passed through (and been modified on) different routes through the system. So we obtain a natural model of data flow in distributed systems. The idea is to view a feature structure as representation of partial information.

In the RENEW tool [1], the FS Net formalism has been included (undocumented) since release 1.6. The FS Net mode comes with a tool to define a type system by drawing a hierarchy of concepts (concept system). Hence the whole system model can be created inside RENEW. This paper is based on parts of the dissertation of Frank Wienberg [3] (in German).

2 Informal Introduction to Types, Subsumption, and Feature Structures

This section introduces the basic concepts of types, subsumption, and feature structures. Due to space limitations we cannot quote the full definitions here, but we give an idea of what they express. We mostly follow Carpenter [2], unless otherwise stated, so that it should be easy to find the original definitions.

2.1 Types and Subsumption

Types are used to define the attributes that are allowed for entities and the types of (the values of) these attributes. Instead of “attribute” the term “feature” is used. A sub-type inherits the feature types of its super type whereby the co-domain of the feature may be restricted. We say that the super type then *subsumes* the sub-type. The subsumption relation has to be a closed partial order, so that we can define the *join* of a set of types as the least general type that is more special than any other type of the set.

The type system is the fundamental concept for the mathematical definition of feature structures. In the “real world” use the handling of type systems is impractical. Therefore, Wienberg introduces in [3] the notion of a concept system. The use of a concept system instead of a type system has the advantage of relaxed consistency requirements. It is possible to generate a type system for a given concept system, if the latter complies with certain conditions.

Fig. 1 shows the UML concept system from which the type system for all feature structures in Sections 2.2 and 3 are derived. The derived type system differs from the visible concept system in some details. Most important is the additional type \top , and a common subtype *Left/Right* for the overlapping inheritance defined by the distinct arrows below the type *Orientation*.

2.2 Feature Structures

Formally, a feature structure is based on a type system and comprises a finite set of *nodes* with exactly one *root node*, a *node typing function* that maps the nodes to types and a partial *feature node function* that maps pairs of nodes and features to nodes. Usually, different feature structures on a common type system are considered, so that the latter may be omitted.

Feature Structures may be seen as directed graphs with a distinct root node (see right hand side of Fig. 2). Each node is labeled with a type, and arcs are labeled with features. An alternative, more compact representation defined in [2] is the attribute-value-matrix (AVM) notation that emphasizes on the structure as nested records. Wienberg merges in [3] the AVM notation with the notation of UML class and object diagrams. This notation emphasizes on the character of feature structures as descriptions of objects, so that they resemble to UML object views in a collaboration diagram.

As we use the AVM notation in the rest of the paper, we explain it using the example that is shown in the left hand side of Fig. 2. The figure shows a feature structure that represents a pair of socks that share the same color. It is based on the type system derived from Fig. 1.

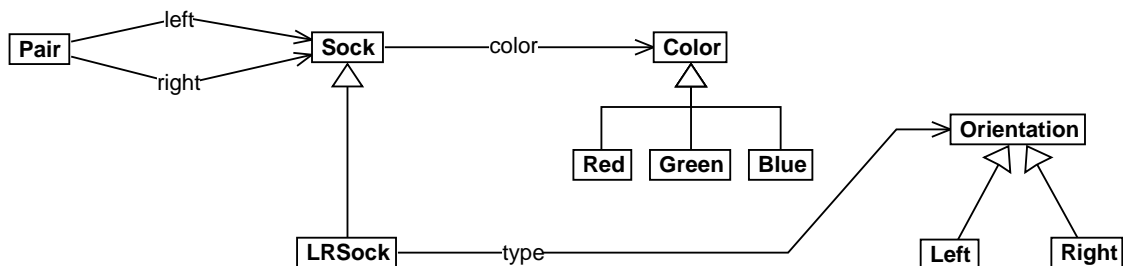


Fig. 1. The concept system on which the following FSNetS are based.

The AVM-like notation of a feature structure consists of nested bracket pairs, each bracket pair representing a node of the graph along with its features (attributes). The numeric identifiers in the small boxes are used to identify node references in the graph that cross nested record boundaries. All boxes with the same identifier refer to one identical node. The scope of each name extends all over the outermost box of the feature structure.

For the modeler's convenience, some shorthands in the notation of feature structures in AVM are defined: If all information for a referenced node is given at another location in the figure, we can omit everything except the node identifier. For example, in Fig. 2, the fact that the node named $\boxed{1}$ is of type **Red** is expressed only once. If the type of a feature does not differ from the defined feature type in the type system, it may be omitted.

The concept of subsumption may be carried forward from types to feature structures in the following way: a feature structure F subsumes a feature structure F' , if F' contains all the information from F and possibly some additional information. There are three possibilities to add information to a given feature structure: (1) by adding a feature (an arc in the graph) and possibly a new value (a new node), (2) by refining the type of a given value (node), and (3) by identifying values (nodes are merged).

As in the case of subsumption there is an extension of the join to feature structures. This extended join is called *unification*. The unification of two feature structures yields the most general feature structure that subsumes the two unified ones. The join is not necessarily defined for all types, thus the unification of two feature structures may also be undefined. We further restrict unification to *well-typed unification*, where only such feature structures are allowed where each node type is subsumed by all types of the feature typings of the features that lead to this node.

Subsumption is useful, for example, for queries on a set of data objects. Only those data objects should be chosen, that are subsumed by the (under-specified) query data object. Unification, on the other hand, is useful for consistency checks of data objects and for the combination of consistent data objects. Feature structures may be used as description for data objects, be it under-specified or not.

3 Basic Feature Structure Nets

The standard notation for FS Nets includes some shortcuts convenient for the use as a modeling language. The mapping onto the formal definition of Basic FS Nets is given in the original work of Wienberg [3]. Fig. 3 shows a net that takes two individual socks that then form a matching pair of socks. It is based upon the type system derived from the concept system presented in Fig. 1.

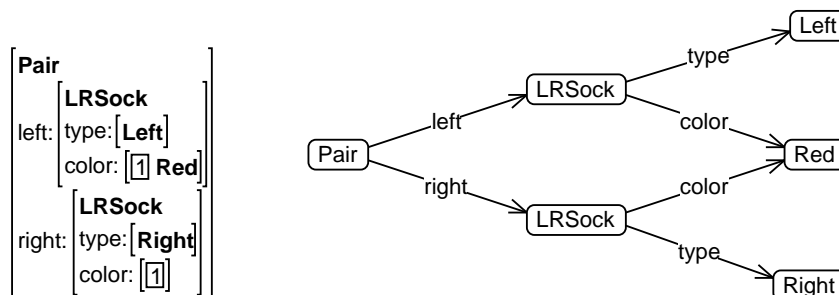


Fig. 2. Two alternative representations for a feature structure.

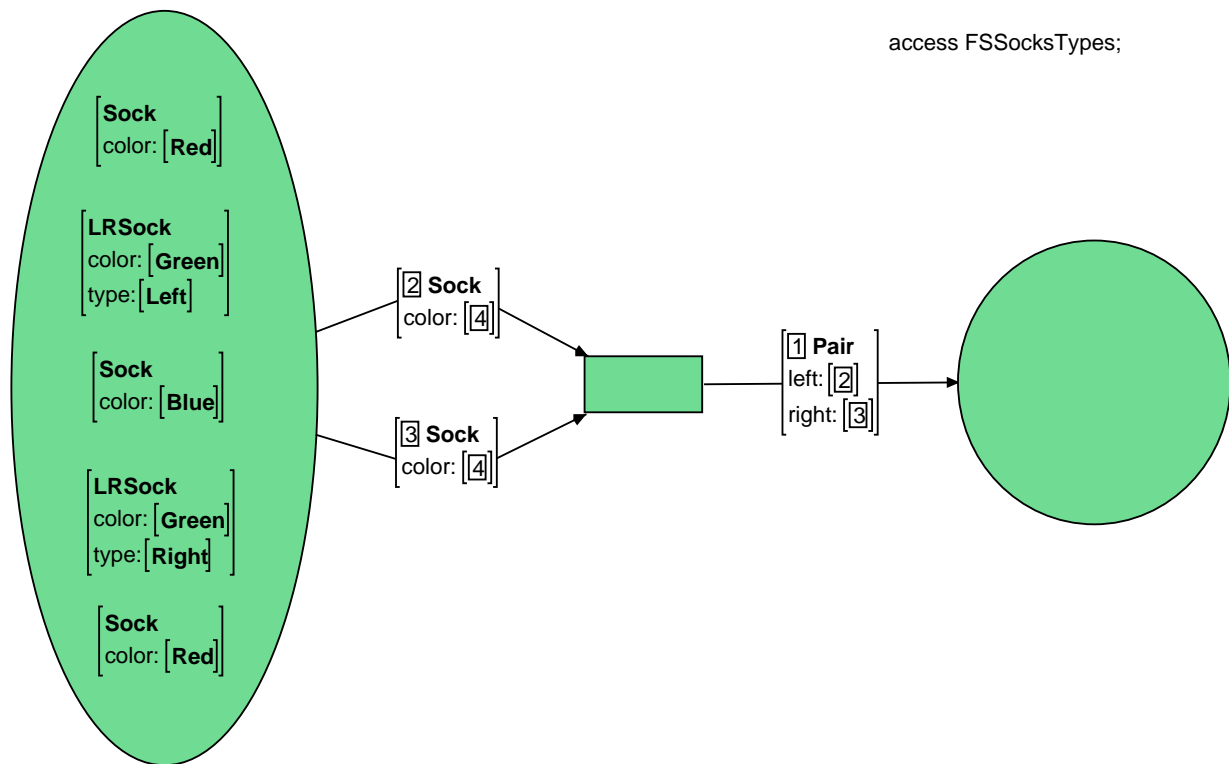


Fig. 3. Example net using the standard notation.

All arcs, places and transitions can be inscribed using feature structures. For places, two kinds of inscriptions are distinguished: type inscriptions and initial marking inscriptions. Type inscriptions restrict tokens of that place to feature structures of the given type. However, in Fig. 3, type inscriptions are omitted. Initial marking inscriptions are located inside the places. In the example only one of the three places, namely $p1$, is initially marked with several feature structures that each represent an individual sock.

Arc and transition inscriptions are not interpreted alone, but in a combination of all arc and transition inscriptions in the locality of each transition. As a consequence, the scope of node identifiers also extends across all feature structures in the locality of a transition. In the example in standard notation, the transition has two input arcs each inscribed with a feature structure of the type $[Sock]$ with a node identifier of $[2]$ resp. $[3]$. It also has an output arc inscribed with a feature structure of the type $[Pair]$ where the features $left$ and $right$ also refer to $[2]$ and $[3]$, respectively. The effect is that the two feature structure tokens coming from $p1$ are unified into the feature structure denoted on the output arc, and the result is put into the place $p2$. The node identifier $[4]$ for the $color$ feature of both input feature structure ensures that the pair will comprise socks of matching color. This comes close to a declarative programming style like in Prolog.

It should be noted that transition and arc inscriptions are somewhat interchangeable in the standard FSNet notation. A modeler could choose to move some or all of the additional node information that is given in the transition inscriptions to the arc inscriptions – it would not make any semantic difference. So the modeler can separate the information as he or she likes, depending on the aspects of the model (information structure and flow) he or she wants to emphasize on.

To give a simple operational semantics for Basic-FSNets, we define the Petri net concepts of activation and firing of a transition. Thereby we obtain interleaving semantics for Basic-FSNets, that allow only for restricted discussion of concurrency and causality of modeled systems. In [3], the restricted class of elementary FSNets is defined along with step and process semantics that allow full concurrency.

In colored nets, one uses the term of firing mode or color (in analogy to the “color” of tokens) when it comes to activation and firing of a transition. The definition of the firing mode is based on the concept of a variable binding. In FSNets we do not have any variables, but the firing mode can be represented naturally by a feature structure. In our example from Fig.3, we just join all inscriptions in the transition’s locality into one feature structure. As there is no transition inscription, and as the input arc inscriptions are unified within the output feature structure, the overall transition firing rule can be represented by a feature structure of type [Pair]. The actual firing mode can be given as a feature structure that specializes the transition rule by exactly the information given by the token binding. The successor marking for an out place is obtained by extraction of the node from the firing mode that corresponds to the out arc inscription.

An important difference between activation and firing of FSNets and other high-level Petri net formalisms lies in the use of equality and identity. All other formalisms (to our knowledge) have the property that identical arc inscriptions correspond to the flow of equal tokens. In FSNets, this property holds for output arcs only. Transitions in FSNets unify existing information, therefore tokens bound to input arcs with identical inscriptions need not be the same, they merely need to be unifiable. The unification of input tokens during a firing corresponds to the idea of a feature structure as representation of partial information. Information in a feature structure that has not been declared explicitly as negative (see Chapter 7 in [2]) is assumed as unknown.

The Basic-FSNet formalism provides a compact representation for the unification of information analogously to the control flow. Without the need for any arc inscription, a transition can remove tokens from multiple input places, unify the information and copy the result to multiple output places. However, a caveat is included in this notation: unless one defines special reserve or test arcs (as is done with the definition of higher FSNets in [3]), a token in a place that is designed as side condition (i.e. connected by an input and an output arc with identical inscriptions) is in fact replaced by a unified copy that may differ from the original token.

4 Discussion and Outlook

The FSNet modeling technique builds upon mainly two well-known techniques: UML and (colored) Petri nets.

In the area of business process modeling, Oberweis uses Prolog-Nets (see [12]) in the tool Income. In Prolog-Nets, a transition rule is given declaratively (in contrast to many other high-level Petri net formalisms where the computation of output tokens is specified operationally). This is a common property with FSNets. In fact, Carpenter shows in [2] that logic programming can be simulated by feature structures.

Weitz proposes in [13] high-level nets that use SGML to represent structured document types. A straight transfer from SGML to XML has been done in the XML nets as presented by Lenz and Oberweis in [14]. Both formalisms come without a full formal specification. A speciality of both formalisms is that input and output arcs can access parts of a token, so that the contents of the token is modified while the token remains on its place. This is unusual for Petri net formalisms, because it complicates the mapping onto uncolored nets.

As mentioned in the introduction, this paper is based on parts of the dissertation of Frank Wienberg [3]. In his work, he presents more variants of FSNeTs. Starting with the basic and higher FSNet formalisms that are documented in this paper, Wienberg also defines elementary FSNeTs that restrict the formalism, defines process semantics, discusses value and reference semantics and copes with the question of universal FSNeTs.

Wienberg shows in a case study that FSNeTs are well suited for the modeling of workflows. Another possible application area for FSNeTs are Multi-agent systems (MAS). Wienberg shows in his dissertation how to model a knowledge base of a BDI agent using FSNeTs. When it comes to agent communication, the key-value-tuple concept of FIPA ACL messages (see [16]) looks very similar to feature structures. So the integration of FSNeTs in our RENEW-based MAS implementation Mulan/Capa (see [17]) might be useful.

References

1. Kummer, O., Wienberg, F., Duvigneau, M.: Renew – the reference net workshop. (Online: <http://www.renew.de/>)
2. Carpenter, B.: The Logic of Typed Feature Structures: with applications to unification grammars, logic programs and constraint resolution. Number 32 in Cambridge tracts in theoretical computer science. Cambridge University Press, New York (1992)
3. Wienberg, F.: Informations- und prozessorientierte Modellierung verteilter Systeme auf der Basis von Feature-Structure-Netzen. Dissertation, Universität Hamburg, Fachbereich Informatik (2001)
4. Valk, R.: Petri Nets as Token Objects - An Introduction to Elementary Object Nets. In Desel, J., Silva, M., eds.: ICATPN'98, Lisbon, Portugal. Number 1420 in LNCS, Berlin, Springer-Verlag (1998) 1–25
5. Kummer, O.: Referenznetze. Logos-Verlag, Berlin (2002)
6. S. Jablonski et.al., ed.: Workflow-Mangement: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie. dpunkt.verlag (1997)
7. Blackwell, A.: Metacognitive theories of visual programming: What do we think we are doing? In: Proceedings IEEE Symposium on Visual Languages. (1996) 240–246
8. Moldt, D.: Höhere Petrinetze als Grundlage für Systemspezifikationen. Dissertation, Universität Hamburg, Fachbereich Informatik (1996)
9. Kennedy, A.: An overview of the xUML system development process. Technical report, Kennedy Carter, Guildford, UK (1999)
10. Boger, M., Baier, T., Wienberg, F., Lamersdorf, W.: Extreme modeling. In: Extreme Programming and Flexible Processes in Software Engineering - XP2000, Addison Wesley (2000)
11. Jensen, K.: Coloured Petri Nets: Volume 1; Basic Concepts, Analysis Methods and Practical Use. EATCS Monographs on Theoretical Computer Science. Springer-Verlag (1992)
12. Oberweis, A., Scherrer, G., Stucky, W.: INCOME/STAR – methodology and tools for the development of distributed information systems. Information Systems **19**(8) (1994) 643–660
13. Weitz, W.: Workflow modelling for internet-based commerce – an approach based on high-level petri nets. (1998) 166–178
14. Lenz, K., Oberweis, A.: Inter-organizational business process management with XML nets. In H. Ehrig et.al., ed.: Petri Net Technology for Communication-Based Systems. Volume 2472 of LNCS. Springer-Verlag (2003) 243–263
15. Jacob, T., Kummer, O., Moldt, D., Ultes-Nitsche, U.: Implementation of Workflow Systems using Reference Nets – Security and Operability Aspects. In K.Jensen, ed.: CPN Workshop Proceedings, Aarhus University (2002) DAIMI PB 560.
16. Foundation for Intelligent Physical Agents: FIPA Abstract Architecture Specification. (2002) <http://www.fipa.org/specs/fipa00001/>.
17. Duvigneau, M., Moldt, D., Rölke, H.: Concurrent architecture for a multi-agent platform. In F. Giunchiglia et.al., ed.: Agent-Oriented Software Engineering III. Workshop, Revised Papers. Volume 1420 of LNCS., Springer-Verlag (2003) 59–72