A Janus-Faced Net Component for the Prototyping of Open Systems

Matthias Wester-Ebbinghaus and Daniel Moldt

University of Hamburg, Department of Informatics
Vogt-Kölln-Straße 30, D-22527 Hamburg
`http://www.informatik.uni-hamburg.de/TGI`

Abstract. We introduce a Janus-faced reference net component that presents the basis for the recursive composition of complex systems from open system units. We particularly focus on the operational aspect of relating different levels of action at different system levels.

1 Introduction

We have presented various aspects of our organization-oriented software engineering approach Sonar/Organ [1–4] in previous contributions. Sonar (**S**elf-**O**rganizing **N**et **AR**chitecture) focusses on an exact mathematical body of rules and regulations for activities in an organizational position network. Orthgonally, Organ (**ORG**anizational **A**rchitecture **N**ets) provides a qualitative comprehension model for distinguishing different system levels according to distinctions between the (collective) organizational units studied at each level. In this paper, we focus on operationalizing the most central concept of Organ, namely building systems in terms of modular Janus-faced system units. This may be regarded as a prototypical basis that is open for incorporation of the Sonar rule set and orchestration according to the Organ architectural guidelines.

Organ rests on the universal model of an open and controlled system unit from Figure 1 that is applied at *all* system levels. We distinguish different internal system units (that are again instances of the universal model from Figure 1). Integration units together with operational units represent the “here and now” of the system unit in focus. The operational units are so to say the intrinsic units and carry out the system’s primary activities. They are dependent on the integration units which offer a *technical frame* via intermediary, regulation, and optimisation services in the course of integration processes. The governance units represent the “there and then” of the system unit in focus. They offer a *strategic frame* via goal/strategy setting, boundary management, and transmitting their decisions to the other internal system units in the course of governance processes.

Each system unit is a *Janus-faced* entity. It “looks inwards” by embedding other system units and at the same time “looks outwards” by being itself embedded in other system units. Thus, besides the already mentioned internal (technical and strategic) frames, each system unit in focus is *externally framed* by surrounding system units to which the system unit in focus relates via periphery processes. To conclude, we take a recursive, self-similar nesting approach,

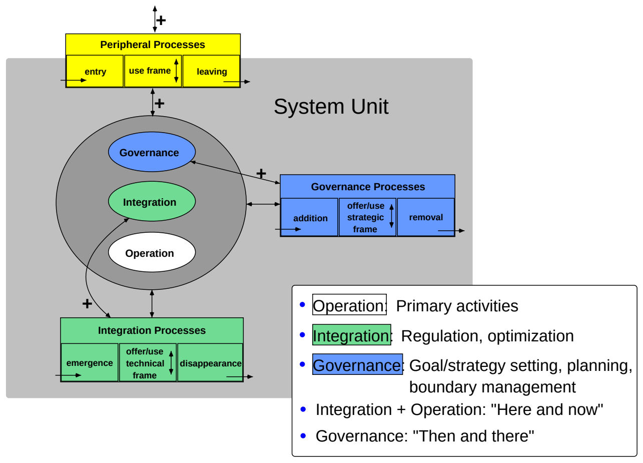
+
Peripheral Processes
entry
use frame
leaving
+
System Unit
Governance
Integration
Operation
+
Governance Processes
addition
offer/use strategic frame
removal
+
Integration Processes
emergence
offer/use technical frame
disappearance
• Operation: Primary activities
• Integration: Regulation, optimization
• Governance: Goal/strategy setting, planning, boundary management
• Integration + Operation: "Here and now"
• Governance: "Then and there"

Fig. 1. Universal Model of an Open and Controlled System Unit

borrowed from Koestler's concept of a *holon* [5] that we extend with a generic reference model for control structures at each level. Thus, we arrive at a modular approach to comprehend *systems of systems*. Each system part may be regarded under a platform perspective and under a corporate agency perspective. All in all, this provides a conceptual basis to systematically study and implement different modes of coupling, both horizontally and vertically. In particular, we have conceptualised a reference architecture for multi-organization systems that exhibits four system levels: the departmental level, the organizational level, the level of organizational fields and the societal level.

In our previous contributions we have stressed the underlying Petri net semantics (specifically, reference net semantics [6]) of the model from Figure 1. However, the model remains rather abstract and we have omitted any real operationalisation details so far. In this paper, we have a look at one particular aspect of Petri net operationalisation in this context. We leave aside any details concerning a qualitative distinction between different system units and processes, how exactly system processes come into being and the addition, removal, migration or expansion of system units. Instead, we focus on one possible technical realization of relating different levels of action at different system levels with each other. We present a recursive approach that allows us to prototype open systems in a modular way, namely by addressing each system level and system unit in turn. Vertical as well as horizontal ties between system units are established via their inclusion in system processes through customisable generic interfaces.

2 Janus-Faced Net Component

Figure 2 displays a reference net component that reduces the various concepts from Figure 1 to internal system units, system processes and internal or peripheral actions of system units in the course of these processes. The component has the already mentioned Janus-faced character: It “looks inwards” by providing a platform for its embedded system units and at the same time “looks outwards” by being itself embedded inside other enclosing system units.

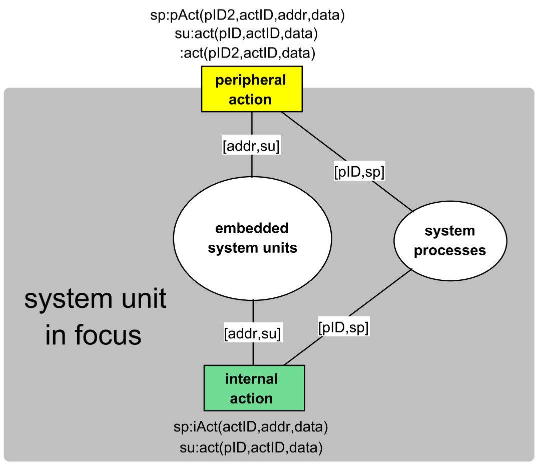

Fig. 2. Janus-Faced Net Component

Each internal action takes place in the course of some system process (sp) and is carried out by some internal system unit (su). It carries an action identifier (actID) that is unique in the context of the associated process. The corresponding system unit is identified by its address (addr) and is passed the identifier of the process (pID) in whose course the action is carried out. Finally, each action is associated with some data (data).

Peripheral actions are also carried out by internal system units. However, from the perspective of surrounding system units, they are carried out by the system unit in focus as a whole. Thus, we arrive at a technical understanding of *collective/corporate* action. As an additional argument, peripheral actions are not only associated with an internal system process but also with the corresponding identifier of the system process of the surrounding system unit (pID2).

We first have a look at a simple production process in Figure 3 with only internal actions. It can be read according to the common UML understanding: Places and Transitions on a vertical line represent the *life line* of some role while places and transitions on a horizontal line represent a *message exchange* between different role players. As we do not look at process instantiation here, we assume that the process net exists permanently (on the place system processes) and may be used for multiple concrete production scenarios. System unit addresses are associated with process roles and product identifiers dynamically at the first transitions of the corresponding life lines.

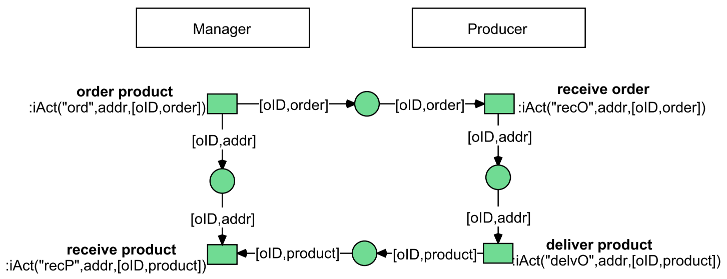

Fig. 3. Simple Production Process

3 Open System Prototyping

We now take a look at open systems by extending our modelling from the previous subsection with an additional system level that brings with it a collective level of action as can be seen in Figure 4. We have to decide, which are the *atoms*

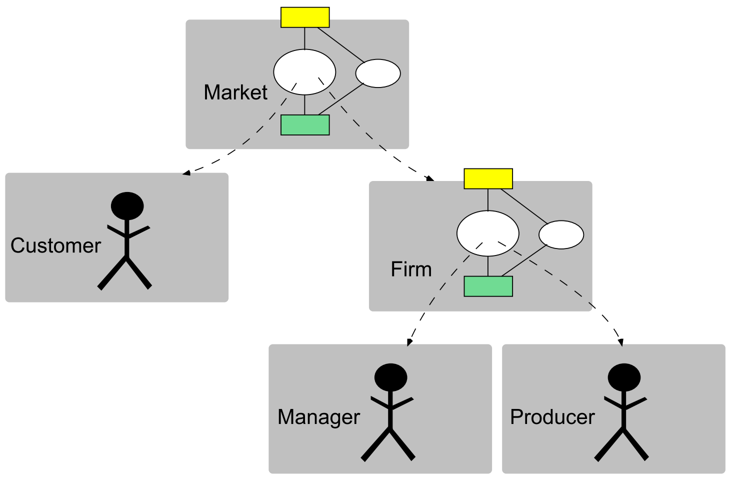

Fig. 4. System with Different Levels of Action

of our system, those parts from which all other activities eventually stem. These atoms are marked in Figure 4 as individuals. As they embed no internal system units themselves, they are not built according to the Janus-faced system unit from Figure 2.

We now have a look at how different levels of action interfere. As an example we take a look at the manager from the production firm that acts in turn on the market. Figure 5 displays how actions of the manager either only take place on the level of the firm or are additionally transformed to collective level actions of the firm itself on the market.

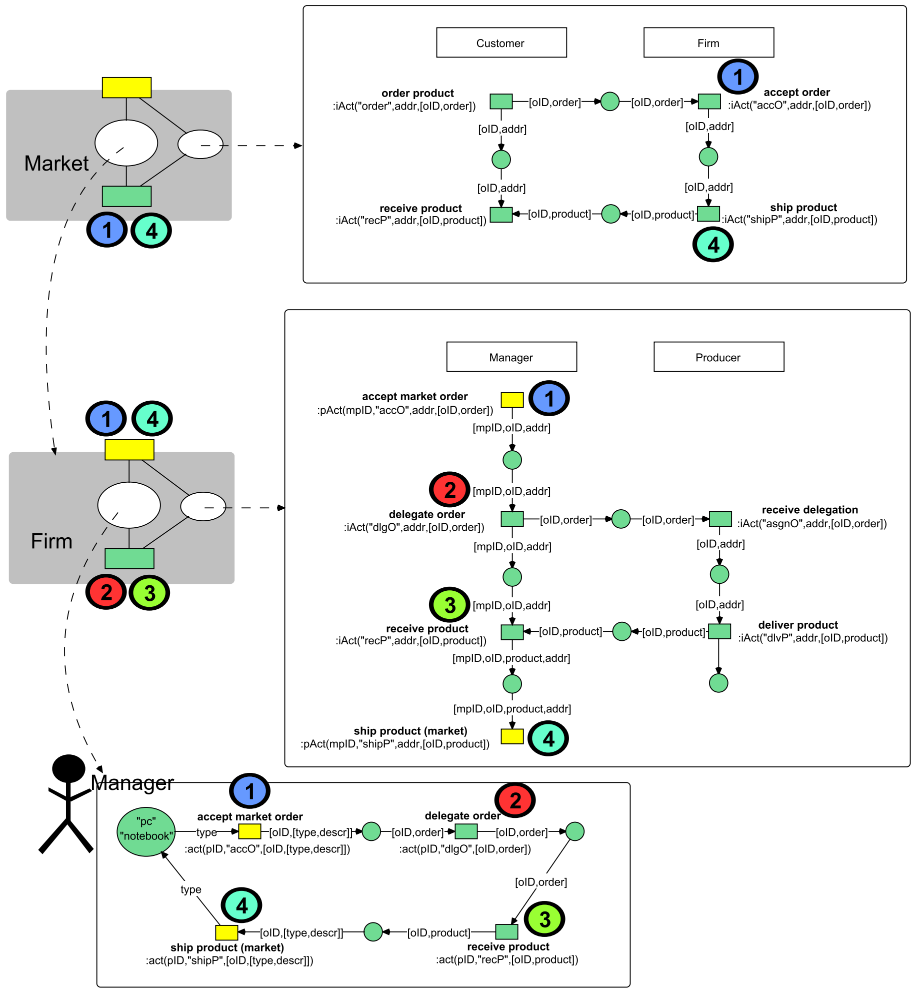
Market
Customer
Firm
order product
:iAct("order",addr,[oID,order])
[oID,order]
accept order
:iAct("accO",addr,[oID,order])
[oID,addr]
receive product
:iAct("recP",addr,[oID,product])
[oID,product]
ship product
:iAct("shipP",addr,[oID,product])
Firm
Manager
Producer
accept market order
:pAct(mpID,"accO",addr,[oID,order])
[mpID,oID,addr]
delegate order
:iAct("dlgO",addr,[oID,order])
receive delegation
:iAct("asgnO",addr,[oID,order])
receive product
:iAct("recP",addr,[oID,product])
deliver product
:iAct("dlvP",addr,[oID,product])
[mpID,oID,product,addr]
ship product (market)
:pAct(mpID,"shipP",addr,[oID,product])
Manager
"pc"
"notebook"
type
accept market order
:act(pID,"accO",[oID,[type,descr]])
[oID,[type,descr]]
delegate order
:act(pID,"dlgO",[oID,order])
ship product (market)
:act(pID,"shipP",[oID,[type,descr]])
receive product
:act(pID,"recP",[oID,product])
1
2
3
4

Fig. 5. Interplay between Different Levels of Action

– **Step 1:** The manager accepts a product order that occurs on the market. The manager accepts on behalf of the firm. Consequently, it acts on two system levels simultaneously: (1) at the firm level where a new production activity is initiated and (2) at the market level where the manager's action is transformed into a collective level action of the firm.

– **Steps 2 + 3:** The simple production scenario from Figure 3 can be found (slightly modified) as a substructure of the system process at the firm level in Figure 5. It still contains only (firm-)internal actions and in steps 2 and 3 the manager delegates the market order to some producer of the firm and afterwards receives the finished product.
– **Step 4:** The manager ships the finished product to the customer on the market. Just like in step 1, the manager acts both on the firm level (itself) and on the market level (its action transformed into a collective level one on behalf of the firm).

4 Outlook

We have presented a reference net operationalisation for combining different levels of action in complex systems that are hierarchically built up from open system units. This addresses one particular aspect of our previously published ORGAN-model for building software systems in the large as systems of systems.

Our operational approach rests on a generic Janus-faced net component that allows to recursively nest system levels inside each other. The generic interface of the component allows to regard each system unit without the need to bother with internal details of lower-level system units. For instance, in the example from Figure 5 it would easily be possible to expand the manager part into a complex system unit containing multiple managers (e.g. a salesman for market exchange management and a foreman for internal production management). This move would have no effect on the current process definition at the firm level. Consequently, our approach allows for the modular prototyping of hierarchically organized systems of systems that are composed of open system units.

References

1. Wester-Ebbinghaus, M., Moldt, D.: Structure in threes: Modelling organization-oriented software architectures built upon multi-agent systems. In: Proceedings of the 7th International Conference an Autonomous Agents and Multi-Agent Systems (AAMAS'2008). (2008) 1307–1311
2. Köhler, M.: A formal model of multi-agent organisations. Fundamenta Informaticae **79**(3–4) (2006) 415–430
3. Köhler, M., Wester-Ebbinghaus, M.: Closing the gap between organizational models and multi-agent system deployment. In: Multi-Agent Systems and Applications V. Volume 4696 of Lecture Notes in Computer Science., Springer-Verlag (2007) 307–309
4. Wester-Ebbinghaus, M., Moldt, D., Köhler, M.: From multi-agent to multi-organization systems: Utilizing middleware approaches. To appear, accepted paper for the 9th International Workshop on Engineering Societies in the Agents World (ESAW'2008) (2008)
5. Koestler, A.: The Ghost in the Machine. Henry Regnery Co. (1967)
6. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)