A New Formal IDEF-based Modelling of Business Processes

Costin Bădică[1], Amelia Bădică[2], and Valentin Liţoiu[2]

[1] Software Engineering Department, University of Craiova
1100 Craiova, Romania,
Email: c_badica@hotmail.com
[2] Business Information Systems Department, University of Craiova
1100 Craiova, Romania

Abstract. This paper formalizes the notation for business processes that has been developed in the INSPIRE project. The process dimension of the notation integrates in a novel way aspects from IDEF0 and IDEF3. The first part of the paper gives an informal introduction of this notation. Then the abstract syntax is described in a formal way using set theory. The formalization allowed to capture concisely intuitive notions like activity decomposition and level of detail of a business process model and to prove two important results that are consistent with our intuition: i) "composing" activity decompositions yields a more refined decomposition and ii) any level of detail of a business process model is a decomposition of the root activity. The process dynamics is formally interpreted by mapping a level of detail to a Place/Transition net. The mapping is described by an algorithm which is linear in the size of the process model.

1 Introduction

An increased interest in applying information technology to business processes has been manifested during the last decade. Because organizations are very complex artifacts, it has been claimed that carefully developed models are necessary for describing, analyzing and/or enacting their underlying business processes ([1]). A business process model is expressed with a graphical notation able to capture all the relevant model information and additionally to facilitate the model analysis by static verification and/or dynamic simulation.

The aim of the INSPIRE project (IST-10387-1999) was to develop an integrated tool-set to support a systematic and more human-oriented approach to business process re-engineering (BPR hereafter). According to INSPIRE, BPR is a structured approach towards changing the current situation - As Is, to a new situation - To Be, with the goal to radically revising it in order to accommodate new organizational needs and achieve some benefits. An INSPIRE BPR process is defined as a sequence of stages: understanding and capturing the current situation as an As Is, search for a better solution, i.e one or more improved To Be processes, implementation, i.e. produce an implementation plan and then run it and finally monitor and evaluate the results. During all these stages the modeling and analysis activities of business process are very important. Therefore, a central aspect was the development of a notation for representing business processes that is both easy to use and understand by the project partners and sufficiently rigorous to allow static verification and dynamic simulation ([2]). The process dimension of this notation is based on IDEF0 ([3]) and enhanced with facilities for representing the dynamics of business processes from IDEF3 ([4]).

The paper is structured as follows: section II gives an informal introduction of the notation and the rationale behind it; section III describes the syntax of INSPIRE models in a formal way; section IV describes the mapping of the INSPIRE notation to Place/Transition nets (P/T nets hereafter); section V provides some pointers to related work; section VI concludes the paper.

2 A Notation for Business Processes

2.1 The Black Box Model

IDEF0 is a technique for producing a function model of a new or existing system or subject area ([3]).

The modelling elements of IDEF0 are (i) boxes and (ii) arrows. Boxes represent functions defined as activities, processes or transformations and arrows represent data or objects related to functions. A box describes what happens in a designated function. In IDEF0 a box has attached to it four types of arrows: inputs, outputs, controls and mechanisms. Mechanism arrows are further subdivided into resource or support arrows and call arrows. According to [3], inputs are transformed or consumed by the the function, outputs are the data or objects produced by the function, controls specify the conditions required for the function to produce correct outputs, mechanism arrows pointing upwards identify some means for supporting the execution of the function, and mechanism arrows pointing downwards are call arrows enabling the sharing of details between models.

One goal of the INSPIRE notation was to retain as much as possible from IDEF0 and to avoid or constrain those aspects that were considered problematic. Because the focus in INSPIRE was on re-engineering and because dynamic simulation was considered a very important issue, a special attention has been paid to produce a function modeling component that allows a neat extension to incorporate the modeling of dynamic aspects.

An INSPIRE activity has input flows, output flows, control flows and mechanism flows (see figure 1). Controls were considered a special kind of inputs, with a particular interpretation for modeling dynamic aspects, even if this might look too restrictive. A control expresses a condition that must be true in order for the activity to start. In this way the goal of eliminating ambiguities from IDEF0 has been achieved with the price of reducing the expressivity. The mechanisms bring to an activity the resources, either humans or machines, i.e. agents, that are needed to execute it. An important distinction has been drawn between inputs, outputs and controls on one side and mechanisms on the other side. The first are called *data flows* because their purpose is to carry data or objects to or from activities. Mechanisms are called *resource flows* because their purpose is to bring resources to activities.

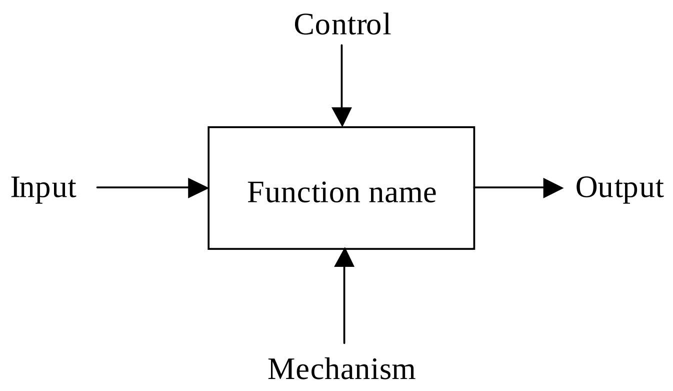

Fig. 1. An INSPIRE activity

A special attention has been paid to branching arrows (forks and joins, known also as bundling arrows in IDEF0). Branching on data flows were considered separately from branching on resource flows.

A fork is an arrow from source to use that is divided into two or more branching arrows ([3]). Because IDEF0 does not impose any constraints on the dynamics and because unconstrained forks on data flows were considered a source of confusion in dynamics modeling, the following assumption was introduced: a fork on a data flow indicates that the item placed onto the arrow from source to use will be made available to all its destinations. In this case all the arrows will have the same label. Thus distinct labels on the branches of a fork on a data flow are not allowed. If the fork is not intended to model this situation, than it must be replaced with a single input/multiple output activity. The next section shows that forks on data flows are just a form of syntactic sugar of the notation and they are not really needed for theoretical investigation.

A join is a point where two or more arrows from source to use are merged into a single arrow ([3]). Merging is different from splitting because it is difficult to imagine that it could happen outside an activity. Merging was considered a source of confusion even if the labels of the branching arrows are the same as the parent arrow and thus they were completely eliminated from the notation. Whenever is needed to model a join-like situation, it is suggested to use multiple input/single output activities and to clarify the semantics using the dynamics part of the notation.

Only branches of type fork are allowed for mechanisms. The pool of resources or agents needed to execute the activity is attached to a mechanism. Thus it was considered natural to have a compositional semantics for the forks on mechanisms. Two situations must be distinguished. If at least one branch on a mechanism arrow has a label different than the label of the parent arrow then the fork models the fact that some parts (i.e. subsets of resources) of the pool of resources attached to the parent arrow are taken and attached to (some of) the child arrows. If a child arrow is not labeled then it makes sense to assume that it is labeled by default with the label ”inherited” from the parent arrow (i.e. the original IDEF0 interpretation is used) and thus the pool attached to the parent arrow is made available to the child arrow, as well. A small plain triangle symbol was used to indicate the first situation and a small plain circle symbol to indicate the last one (see figure 3). The circle symbol is used on forks on data flows as well.

For example, in a manufacturing company there is a business process for material acquisition. It takes material requests and produces purchase orders and payment authorizations. It contains a sub-process for handling the material requests that takes material requests and produces validated requests. The company has a list of available suppliers, but is must be prepared to find and handle new potential suppliers. So, there is an additional input to handle new supplier requirements and an additional output to produce new supplier packages. Two resource pools were identified for this process: data systems needed to log and track the request, to register the product, a.o. and materials staff including clerks, supervisors, a.o.

The process is represented at an abstract level in figure 2. For the meaning of the activities names and flow names see table 1.

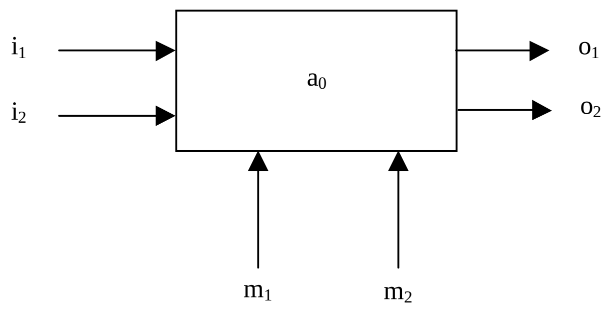

Fig. 2. The top-level abstraction of a business process

The notation allows the presentation of a model at different levels of detail. The process in figure 2 is detailed in figure 3. Figures 2 and 3 are called diagrams. Figure 2 is called the top-level or root diagram of the business process.

2.2 Adding Glass Box Views

The functional part of the notation was enhanced with features for modeling the dynamics by attaching to each black box a glass-box view based on IDEF3.

IDEF3 is a technique for producing a dynamic model of the system ([4]). IDEF3 can produce two views: a process-centered view and an object-centered view. Only the

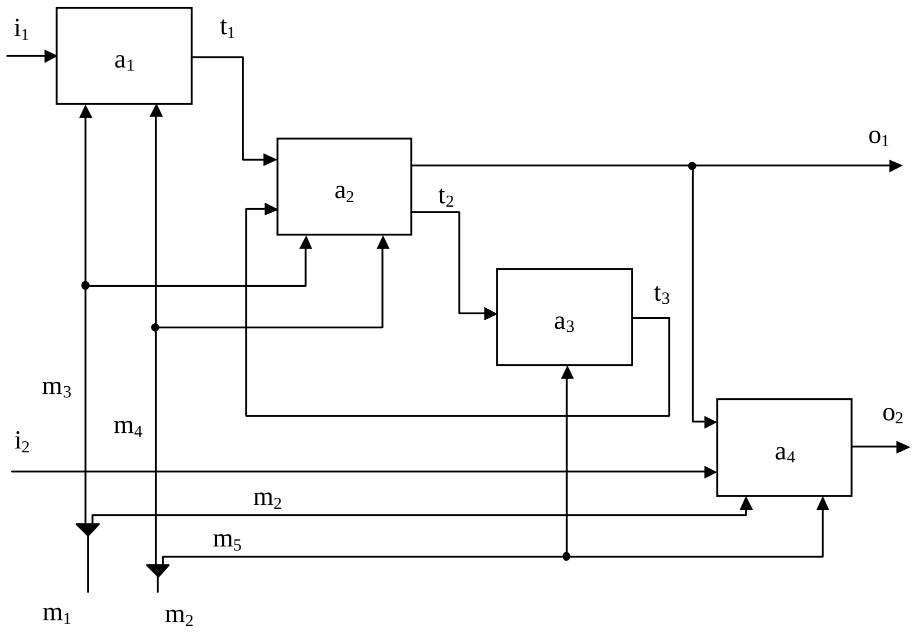

Fig. 3. A more detailed presentation of the process in figure 2

Name	Description
a_0	Process Material Request
a_1	Log Material Request
a_2	Validate Material Request
a_3	Resolve Request Problems
a_4	Develop New Supplier Specification
i_1	Material Request
i_2	New Supplier Requirement
o_1	Validated Request
o_2	New Supplier Package
t_1	Logged Request
t_2	Request Errors
t_3	Request Updates
m_1	Data Systems
m_2	Materials Staff
m_3	Material Tracking System
m_4	Product Data System
m_5	Material Clerk
m_6	Material Supervisor

Table 1. Meanings of symbols used in figures 2 and 3

process-centered view was considered here. The main building blocks of the process-centered view of IDEF3 are (i) units of behavior, (ii) links and (iii) junctions. Units of behavior represent types of happenings (events, acts, etc.), links represent temporal relations between units of behavior, and junctions are used to specify the logic of process branching. Within the INSPIRE tool the term connector is used instead of junction.

A glass-box view contains a unit of behavior, a tree of input connectors and a tree of output connectors. There are one-to-one mappings between the inputs of a black box and the leaves of its input tree and between the outputs of a black box and the leaves of its output tree. The unit of behavior models the "instantiation" of the activity. The input tree models the logic of selecting the inputs participating in the activity, and the output tree models the logic of generating the outputs produced by the activity. The INSPIRE notation allows the use of four connector types: input exclusive or (*ixor*), input and (*iand*), output exclusive or (*oxor*) and output and (*oand*).

Consider again the example in figure 3. Glass-box views are associated to activities a_2 (Validate Material Request) and a_4 (Develop New Supplier Specification). Activity a_2 may take t_1 (Logged Request) or t_3 (Request Updates) and may produce either o_1 (Validated Request) or t_2 (Request Errors). Activity a_4 takes both o_1 (Validated Request) and i_2 (New Supplier Requirement) to produce o_2 (New Supplier Package). This is modeled in figure 4.

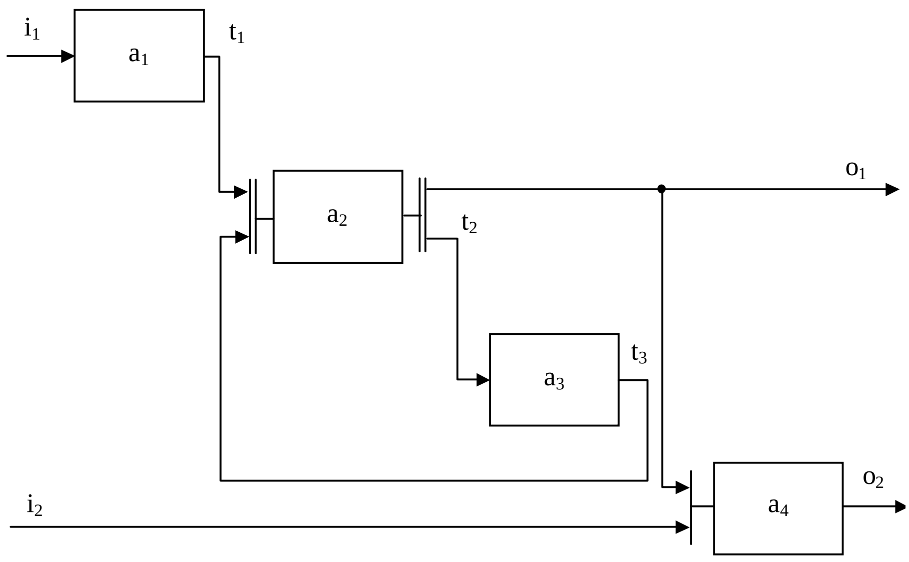

Fig. 4. The result of adding glass-box views to the process in figure 3

Note that the introduction of glass box views simplifies the black-box level of the notation by eliminating forks on data flows. For example, the fork on o_1 can be incorporated into the tree of output connectors of activity a_2 by adding an output *oand* connector. In this way the activity a_2 will have an additional output o'_1.

If a fork occurs on an input interface flow on a diagram, it can be replaced with a single input/multiple output dummy activity with the tree of output connectors consisting of a single output *oand* connector.

The way that controls condition the execution of an activity is modelled by adding a top level *iand* connector that binds together into a single input tree the controls and the tree of input connectors of the activity inputs.

3 Formalizing the INSPIRE Notation

This section presents the INSPIRE notation in a concise and unambiguous way, using set theory. The description is focused on the process dimension of the notation. This formalization allowed to capture precisely the intuitive notions of activity decomposition and level of detail of a business process model and to formally prove two important results: i) the act of "composing" decompositions yields a more refined decomposition (proposition 5) and ii) a level of detail of a business process model is a decomposition of the root activity of the business process (proposition 6). In particular, proposition 6 relates this section, which is focused on the syntax of the notation, with section 4, which is focused on the dynamics of the notation.

3.1 Activities and Diagrams

Let AN be a set of *activity names* and let FN be a set of *flow names*. The set FN is partitioned into the set DFN of *data flow names* and the set RFN of *resource flow names*.

Definition 1. *(connector trees)*

a. *A* tree of input connectors t *and its set* $leaves(t)$ *of* leaves *are defined recursively as follows:*
   i) *If* $f \in DFN$ *then* $t = f$ *is a tree of input connectors and* $leaves(t) = \{f\}$
   ii) *If* Ts *is a nonempty set of trees of input connectors then* $t_1 = iand(Ts)$ *and* $t_2 = ixor(Ts)$ *are tree of input connectors with* $leaves(t_i) = \bigcup_{t \in Ts} leaves(t)$ *for* $i = 1, 2$.

   Tree of output connectors *are defined similarly, except that their roots are labeled with oand and oxor.*
b. *Additionally* undefined tree of connectors *are allowed for the case when glass box views are not attached to activities. They are denoted by* $t_1 = iundef(Fs)$ *and* $t_2 = oundef(Fs)$ *such that* $Fs \subseteq DFN$. *Moreover,* $leaves(t_i) = Fs$, $i = 1, 2$ *are defined.*
c. *The following are defined for a tree of connectors* t*:*
   i) *A predicate* $Leaf(t)$ *that is true iff* t *is a leaf node.*
   ii) *A function* $subtrees(t)$ *that is defined iff* $Leaf(t) = false$ *and returns the set of subtrees of* t.
   iii) *A function* $root(t)$ *that is defined iff* $Leaf(t) = false$ *and returns the root of* t.

An activity has four components: a name, an input tree, an output tree and a set of resource flows called mechanisms.

Definition 2. *(activities)*

a. *An* activity *is a quadruple* (a, it, ot, ms) *such that* $a \in AN$ *is the activity name,* it *is the activity's tree of input connectors,* ot *is the activity's tree of output connectors and* $ms \subseteq RFN$ *is the activity's set of mechanism flow names. Any two distinct activities have distinct names, i.e.* $act_1 \neq act_2$ *iff* $a_1 \neq a_2$*. The following selector functions on activities are defined:* $activityName(act) = a$*;* $inputTree(act) = it$*;* $outputTree(act) = ot$*;* $mecs(act) = ms$*.*
b. *For each activity* act *the* set of inputs, $is(act) = leaves(it)$ *and the* set of outputs, $os(act) = leaves(ot)$ *are defined. It is required that the inputs and outputs of an activity are disjoint, i.e. for all activities* act*,* $is(act) \cap os(act) = \emptyset$*.*

The INSPIRE notation allows to group several activities into a single block called diagram. In addition to a set of activities, a diagram also contains a set of bundled mechanism flows.

Definition 3. *(bundled mechanisms and diagrams)*

a. *A* bundled mechanism *is a pair* $bm = (mec, mecs)$ *such that* $mec \in RFN$*,* $mecs \subseteq RFN$ *and* $mecs \neq \emptyset$*.* mec *is the name of the bundled mechanism and* $mecs$ *are the names of the parts the bundled mechanism is split in. The following selector functions on bundled mechanisms are defined:* $mec(bm) = mec$*;* $mecs(bm) = mecs$*.*
b. *A* diagram *is a pair* $d = (acts, bms)$ *such that* $acts$ *is a set of activities and* bms *is a set of bundled mechanisms. The following selector functions on diagrams are defined:* $acts(d) = acts$ *and* $bms(d) = bms$*. A diagram is required to satisfy the following constraints:*
   i) *Any two distinct activities in a diagram have no inputs in common and no outputs in common, i.e. for all* $act_1, act_2 \in acts$ *if* $act_1 \neq act_2$ *then* $is(act_1) \cap is(act_2) = \emptyset$ *and* $os(act_1) \cap os(act_2) = \emptyset$*.*
   ii) *Each part of a bundled mechanism is connected to at least one mechanism of an activity in the diagram, i.e.* $\bigcup_{bm \in bms(d)} mecs(bm) = \bigcup_{act \in acts(d)} mecs(act)$*.*
   iii) *For all* $bm_1, bm_2 \in bms$ *if* $bm_1 \neq bm_2$ *then* $(mecs(bm_1) \cup \{mec(bm_1)\}) \cap (mecs(bm_2) \cup \{mec(bm_2)\}) = \emptyset$*.*

Part a of definition 3 allows to have $mec(bm) \in mecs(bm)$. Let us assume that $mec(bm) = \{m_0\}$ and $mecs(bm) = \{m_0, m_1, \ldots, m_k\}$. Then $m_1, \ldots m_k$ are interpreted as strict parts of the "whole" m_0. The fact that m_0 is listed among its own parts means that there is at least one activity in the diagram that requires the whole bundle m_0 of resources rather that just one or more of its strict parts.

Condition i) of part b of definition 3 assumes that the forks on the data flows in the diagram have been quietly removed as stated in section 2, paragraph 2.2.

Condition iii) of part b of definition 3 requires no mixture of the mechanism flows of any two distinct bundled mechanisms on the same diagram.

Definition 4. *(diagram flows) The following sets of flows are defined for a diagram d:*

a. The set of data flows *of d is the union of the inputs and outputs of all activities of d, i.e. $data(d) = \bigcup_{act \in acts(d)} is(act) \cup os(act)$.*
b. The set of input flows *of d consists of the inputs of activities of d that are not outputs of any activity of d, i.e. $inputs(d) = \bigcup_{act \in acts(d)} is(act) \setminus \bigcup_{act \in acts(d)} os(act)$.*
c. The set of output flows *of d consists of the outputs of activities in d that are not inputs of any activity in d, i.e. $inputs(d) = \bigcup_{act \in acts(d)} os(act) \setminus \bigcup_{act \in acts(d)} is(act)$.*
d. The set of private data flows *of d contains all the flows of d that are both input to an activity of d and output to an activity of d, i.e. $private(d) = \bigcup_{act \in acts(d)} is(act) \cap \bigcup_{act \in acts(d)} os(act)$.*
e. The set of bundled mechanism flows *of d is the set of names of all the bundled mechanisms of d i.e. $bmechanisms(d) = \bigcup_{bm \in bms(d)} \{mec(bm)\}$.*
f. The set of mechanism flows *of d is the set of names of all the mechanisms that occur in d i.e. $mechanisms(d) = \bigcup_{bm \in bms(d)} mecs(bm) \cup \{mec(bm)\}$.*
g. The set of private mechanism flows *of d is the set of names of all the mechanisms that occur in d and are strict parts of the bundled mechanisms of d i.e. $pmechanisms(d) = \bigcup_{bm \in bms(d)} (mecs(bm) \setminus \{mec(bm)\})$.*

Proposition 1. *For all diagrams d the family of sets of inputs, outputs and private data flows of d is a partition of the set of data flows of d and the family of sets of bundled mechanism flows and private mechanism flows is a partition of the set of mechanism flows of d.*

3.2 Decompositions

An important concept in INSPIRE is the presentation of a model at different levels of detail, based on the notion of decomposition.

Definition 5. *(decomposition) Let d be a diagram and let $act \notin acts(d)$ be an activity. d is called a* decomposition *of act iff the following conditions hold:*

i) $inputs(d) = is(act)$
ii) $outputs(d) = os(act)$
iii) $bmechanisms(d) = mecs(act)$

Two special cases of diagrams are:

i) $d = (\emptyset, \emptyset)$ called the *empty diagram.*
ii) $d = (\{act\}, \{(mec, \{mec\}) | mec \in mecs(act)\}$ called a *singleton diagram.*

Proposition 2. *The singleton diagram $d = (\{act\}, \{(mec, \{mec\}) | mec \in mecs(act)\}$ is a decomposition of activity act.*

Decompositions can be "composed" to obtain more refined decompositions, following two steps: definition of an operation for composing sets of bundled mechanisms of diagrams and of an operation for removing an activity from a diagram to obtain a new diagram. Composing decompositions is then straightforward. Basically if diagram d_1 is a decomposition of activity act_0, act_1 is an activity of d_1 and d_2 is a decomposition of act_1 then act_1 is first removed from d_1, producing a new diagram d_1^*. Then the composition of d_1 and d_2 is obtained by taking the union of activities of d_1^* and of d_2 and the composition of the sets of bundled mechanisms of d_1^* and d_2. Furthermore, it is proved that the composition of d_1 and d_2 is a decomposition of act_0 as well.

Definition 6. *(composing sets of bundled mechanisms) Let bms_1 and bms_2 be two sets of bundled mechanisms of two diagrams (i.e. they obey condition iii) of part b) of definition 3) such that $(\bigcup_{bm \in bms_1} mecs(bm) \cup \{mec(bm)\}) \cap (\bigcup_{bm \in bm_2}(mecs(bm) \setminus \{mec(bm)\})) = \emptyset$. Then the* composition $bms_1 \oplus bms_2$ *of bms_1 and bms_2 is defined according to the following three rules:*

i) $bms_1 \oplus \emptyset = bms_1$.
ii) $bms_1 \oplus \{bm_2\} =$
1.$(mec(bm_1), (mecs(bm_1) \setminus \{mec(bm_2)\}) \cup mecs(bm_2)$ *if exists $bm_1 \in bms_1$ such that $mec(bm_2) \in mecs(bm_1)$;*
2. $(mec(bm_1), mecs(bm_1) \cup mecs(bm_2))$ *if exists $bm_1 \in bms_1$ such that $mec(bm_1) = mec(bm_2)$;*
3. $bms_1 \cup \{bm_2\}$ *otherwise.*
iii) $bms_1 \oplus bms_2 = (bms_1 \oplus \{bm_2\}) \oplus (bms_2 \setminus \{bm_2\})$ *if $bm_2 \in bms_2$.*

Condition ii) shows that when composing a set of bundled mechanisms with a singleton set of bundled mechanisms three cases must be considered. In the first case bm_2 is decomposing further a mechanism in $mecs(bms_1)$ and thus the new parts must be added to the resulting bundled mechanism. In the second case the "whole" part of the bundled mechanism $mec(bm_1)$ is decomposed in a different way according to bm_2, so the new parts must be collected as well in the resulting mechanism. In case three there is no special interaction between bms_1 and bm_2 so bm_2 is simply added to the resulting bundled mechanism set.

Condition iii) is a recursive definition of composition of sets of bundled mechanisms. It shows how the composition of bms_1 and bms_2 is defined in terms of the composition of bms_1 and a set obtained by removing one element from bms_2.

Proposition 3. *Assuming the conditions of definition 6, the composition $bms_1 \oplus bms_2$ of two sets of bundled mechanisms can be the set of bundled mechanisms of a diagram (i.e it satisfies condition iii) of part b) of definition 3).*

Removing an activity from a diagram means removing it from the set of diagram's activities and updating the set of bundled mechanisms accordingly.

Definition 7. *(removing an activity from a diagram) If $d = (acts, bms)$ is a diagram and $act \in acts$ is an activity of the diagram then the result of removing act from d is a new diagram $d^* = d \ominus act$ defined as $d^* = (act^*, bms^*)$ such that $act^* = acts \setminus \{act\}$ and $bms^* = \{(mec, mecs^*) | (mec, mecs) \in bms,\ mecs^* = mecs \cap \bigcup_{a \in act^*} mecs(a)$ and $mecs^* \neq \emptyset\}$.*

Proposition 4. *d^*, as defined by definition 7, is a proper diagram, i.e. it satisfies the conditions of definition 3, part b).*

The operation for composing decompositions can now be defined.

Definition 8. *(composing decompositions) Let act_0 be an activity, d_1 a decomposition of act_0, $act_1 \in acts(d_1)$ and d_2 a decomposition of act_0 such that $data(d_1) \cap private(d_2) = \emptyset$ and $mechanisms(d_1) \cap pmechanisms(d_2) = \emptyset$. Let d_1^* be the diagram obtained by removing act_1 from d_1, i.e. $d^* = d_1 \ominus act_1$. Then $d = (acts(d^*) \cup acts(d_2), bms(d^*) \oplus bms(d_2)$ is called the* composition *of d_1 and d_2.*

An important property is that by composing decompositions, a proper and more refined decomposition is obtained.

Proposition 5. *d, as defined by definition 8, is a proper decomposition of act_0.*

The interpretation is that if an activity of decomposition is further decomposed in a "clean" way then a more refined decomposition of the initially decomposed activity is obtained.

For all activities *act*, the singleton diagram determined by *act* is a decomposition of *act*. If in definition 8 and proposition 5 d_2 is taken to be the singleton diagram determined by act_1 then the composition of d_1 and d_2 will simply be d_1. So singleton decompositions act as neutral elements for composing decompositions.

3.3 Levels of Detail

To model the tree structure of a business process model at different levels of detail the notion of decomposition structure is needed.

Definition 9. *(decomposition structure) Let $\mathcal{A}$ be a finite set of activities and $\mathcal{D}$ a set of diagrams. A* decomposition structure *rooted at $r \in \mathcal{A}$ is a function $dec : \mathcal{A} \to \mathcal{D}$ such that:*

i) For all $x \neq y \in \mathcal{A}$ if $x \neq y$ then $acts(dec(x)) \cap acts(dec(y)) = \emptyset$.
ii) $\bigcup_{x \in \mathcal{A}} acts(dec(y)) = \mathcal{A} \setminus \{r\}$.
iii) For all $x \in \mathcal{A}$ if $acts(dec(x)) \neq \emptyset$ then $dec(x)$ is a decomposition of x.
iv) For all $x \in \mathcal{A}$ and $y \in acts(dec(x))$, $data(dec(x)) \cap private(dec(y)) = \emptyset$ and $mechanisms(dec(x)) \cap pmechanisms(dec(y)) = \emptyset$.

Conditions i) and ii) ask for the *dec* function to define a tree structure on the set $\mathcal{A}$ of activities. r is the root of this tree and it is called the top level activity. Condition iii) asks for $dec(x)$ to be a decomposition of x. Condition iv) asks for the context of decomposing y to be "clean", i.e. the diagram that contains y should not have any common flows with the private set of flows of the decomposition of y.

A final important concept is the level of detail of a business process model. A business process model in INSPIRE has a tree structure. Clearly, the least refined level of detail of the process is the root of the tree, and the most refined level of detail of the process is the set of leaves. Between them there are many "intermediate" levels of detail. An "intermediate" level of detail corresponds to a cut in the decomposition tree.

Definition 10. *(level of detail) Let $dec : \mathcal{A} \to \mathcal{D}$ be a decomposition structure rooted at $r \in \mathcal{A}$. A* level of detail *in dec is a diagram d defined according to the following rules:*

i) The singleton diagram defined by r is a level of detail in dec.
ii) If l is a level of detail in dec, x is an activity in l, i.e. $x \in acts(l)$ and if $acts(dec(x)) \neq \emptyset$ then the diagram obtained by composing l with $dec(x)$ is a level of detail in dec.

An important result that follows from proposition 5 and definition 10 is that all the levels of detail in a decomposition structure are decompositions of the top level activity.

Proposition 6. *Let $dec : \mathcal{A} \to \mathcal{D}$ be a decomposition structure rooted at $r \in \mathcal{A}$ and l a level of detail in dec. Then l is a decomposition of r.*

4 Mapping the INSPIRE Notation to P/T Nets

The result stated by proposition 6 allows the study of the problem of mapping a level of detail to a P/T net. The resulted P/T net provides the semantics for the presentation of the business process model at a given level of detail. P/T nets have been intensely studied and have a well-established classic theory. They have been used for modelling workflows in [8]. Moreover, there were identified special classes of P/T nets suitable for an efficient static analysis ([14]).

The mapping of the INSPIRE notation to P/T nets does not consider the mechanisms. A similar assumption has also been made in [8] in the context of modeling workflows using P/T nets.

A P/T net is represented using three sets: i) the set of *places*; ii) the set of *transitions*; iii) the set of $arcs \subseteq (places \times transitions) \cup (transitions \times places)$.

The mapping is described by means of an algorithm for translating a level of detail to a P/T net. Assuming that each activity has attached a glass box view, the algorithm will translate each activity tree of input connectors and output connectors. The activity itself is translated into a transition. The flows are translated into places. The translation of trees will produce new places and transitions.

The translation of an activity *act* is given by algorithm *Activity2PTNet*. The translation of a tree *it* of input connectors is given by algorithm *InputTree2PTNet*. The translations of *iand* and *ixor* connectors are given by algorithms *IAnd2PTNet* and *IXor2PTNet*.

```
procedure $Activity2PTNet(act)$
          $l_1 \leftarrow InputTree2PTNet(inputTree(act))$
          $l_2 \leftarrow OutputTree2PTNet(outputTree(act))$
          $t \leftarrow NewTransition(activityName(act))$
          $transitions \leftarrow transitions \cup \{t\}$
          $arcs \leftarrow arcs \cup \{(l_1, t), (t, l_2)\}$
end
```

```
function $InputTree2PTNet(it)$
        if $Leaf(it)$ then
          $r \leftarrow NewPlace(it)$
          $places \leftarrow places \cup \{r\}$
        else
            $trees \leftarrow subtrees(it)$
            $r \leftarrow NewPlace(NewSymbol('l'))$
            $places \leftarrow places \cup \{r\}$
            $rs \leftarrow InputTrees2PTNet(trees)$
            if $Root(it) =' iand'$ then
              $IAnd2PTNet(rs, r)$
            else
                $IXor2PTNet(rs, r)$
        return $r$
end
```

```
procedure $IAnd2PTNet(rs, r)$
   $t \leftarrow NewTrans(NewSymbol('tr'))$
   $transitions \leftarrow transitions \cup \{t\}$
   for each $x \in rs$ do
       $arcs \leftarrow arcs \cup \{(x, t), (t, r)\}$
end
```

```
procedure $IXor2PTNet(rs, r)$
   for each $x \in rs$ do
       $t \leftarrow NewTrans(NewSymbol('tr'))$
       $transitions \leftarrow transitions \cup \{t\}$
       $arcs \leftarrow arcs \cup \{(x, t), (t, r)\}$
end
```

The function $NewSymbol(prefix)$ generates a new symbol with a given prefix. The functions $NewPlace(label)$ and $NewTransition(label)$ generate a new place and a new transition with a given label.

The complexity of the translation algorithm is linear in the number of activities plus the number of connectors plus the number of data flows in the chosen level of detail of the business process.

The result of translating the process in figure 4 is shown in figure 5.

5 Related Work

This paper uses set theory to describe the formal syntax of the INSPIRE notation and the way it combines features from IDEF0 and IDEF3. Then it shows how the dynamics of INSPIRE business processes can be formally understood by mapping the models to P/T nets. Other techniques proposed in the literature for formally understanding business processes are based on flownomial expressions ([5]), process algebras ([6]) or knowledge-based systems ([7]). P/T nets have been used for workflow modeling and verification ([8]).

There are not many references in the literature reporting on the formal analysis of IDEF0 and in particular on relating IDEF0 and Petri nets. Paper [10] discusses how

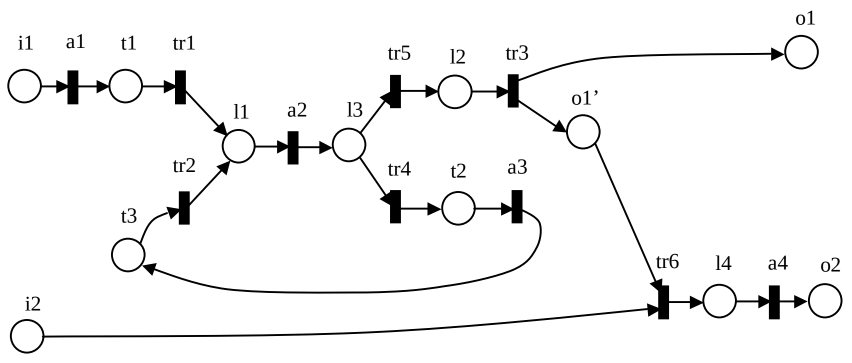

Fig. 5. The result of translating the process in figure 4 to a P/T net

IDEF0 and colored Petri nets can be used for business process modeling. Some clues on how to map IDEF0 diagrams to colored Petri nets are also given in [11]. Paper [9] discusses the suitability and effectiveness of IDEF diagrams (IDEF0 and IDEF3) and Petri nets for modeling business processes in the context of business process re-engineering. A comparative evaluation of their features is also provided there. Also, the INSPIRE notation has some similarities with the formally founded description technique of business processes reported in [12] and [13]. The similarities concern the use of black-boxes, glass-box views and switches (that correspond roughly to the INSPIRE connectors).

6 Conclusions

This paper introduced a notation for business process modeling that incorporates facilities for function modeling from IDEF0 and for dynamic modeling from IDEF3. Those aspects from IDEF0 that were considered problematic have been constrained or eliminated and only aspects from the process-centered view of IDEF3 have been borrowed. In fact, what was obtained is a novel notation for business process modeling that is both easy to use and formal. Its main advantages are: i) the elements of the notation are the well known boxes, arrows and connectors employed by IDEF0 and IDEF3, which are widely used in the practice of business process modeling and ii) the fact that the notation has a meaning underpinned by mathematics (set theory and P/T nets) makes the models suitable for various tasks specific to business process re-engineering like static verification and dynamic simulation.

References

1. Ould M.: *Business Processes: Modeling and Analysis for Re-engineering and Improvement*, John Wiley, (1995)

2. Fox C.: The Process Representation Module (specification), INSPIRE (IST-1999-10387) Deliverable 2.1 (2000)
3. Draft Federal Information Processing Standards Publication 183, Integration Definition for Function Modelling (IDEF0) (1993)
4. Mayer R. et al.: Information Integration for Concurrent Engineering (IICE): IDEF3 Process Description Capture Method Report (1993)
5. Bădică C., Fox C.: Modeling and Verification of Business Processes, *Proceedings IASTED International Conference on Applied Simulation and Modeling*, Crete, Greece (2002) 7-12
6. Schroeder M.: Verification of Business Processes for a Correspondence Handling Center Using CCS, *Proceedings European Symposium on Validation and Verification of Knowledge Based Systems and Components*, Oslo, Norway (1999) 253-264
7. Yu E., Mylopoulos J. J., Lesprance Y.: AI Models for Business Process Re-Engineering, IEEE Expert, 11(4):16-23 (1996)
8. Aalst W.M.P. van der: The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, 8(1):21-66 (1998)
9. Bosilj Vuksic V., Giaglis G.M., Hlupic V.: IDEF Diagrams and Petri Nets for Business Process Modeling: Suitability, Efficacy, and Complementary Use, *Proceedings International Conference on Enterprise Information Systems (IECIS)*, Stafford, UK (2000) 242-247
10. Pinci O., Shapiro R.M.: Work Flow Analysis, Proceedings Winter Simulation Conference, Los Angeles, CA (1993) 1122-1130
11. The Design/CPN Tutorial, `http://www.daimi.au.dk/designCPN/man/Tutorial/`.
12. Thurner V.: A Formally Founded Description for Business Processes. *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE)*, Los Alamitos, CA (1998) 254-261
13. Rumpe B., Thurner V.: Refining Business Processes, *Proceedings 7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Munich, Germany (1998) 205-220
14. Desel J., Esparza J.: Free Choice Petri Nets, *Cambridge Tracts in Theoretical Computer Science*, 40, Cambridge University Press (1995)