Design of Object Processing Systems

Roxana Grigoraş and Cornelis Hoede

Department of Applied Mathematics
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
{grigorasdr,hoede}@math.utwente.nl

Abstract. Object processing systems are met rather often in every day life, in industry, tourism, commerce, etc. When designing such a system, many problems can be posed and considered, depending on the scope and purpose of design. We give here a general approach which involves graph theory, and which can have many applications. The generation of possible designs for an object processing system, known as *synthesis* in the engineering field, is reduced to first solving a graph embedding problem. We believe that our model could be successful and relatively easily implemented in a software tool, called *Smart Synthesis Tool*, so that the engineering design process will perform quicker. We propose three types of graph transformations which aid the way an object processing system can be designed. Future work will show to which extent these transformation types suffice for generating most of the layouts of the object processing systems.

Keywords: Process, design, graph, object
AMS Classification: 05C99

1 Introduction

In many situations objects are to be processed and a design for such a processing system is asked for. In general more designs are possible, called *solutions* of the *synthesis phase*. Synthesis is in this context part of the process where solutions are generated. Synthesis-support tools help the engineers to translate their demands into the optimal realisation form[1]. A solution should satisfy certain requirements, goals, and constraints, boundary conditions.

In many design procedures the first step is to determine a *Process Flow Diagram* (PFD). The object is to be processed in different ways and the various processings are to be carried out in a prescribed order. A PFD is basically a directed graph $\overrightarrow{P}$, with vertices labelled by the type of processing that is to

[1] See http://www.opm.ctw.utwente.nl/research/design_engineering/synthesis/Research.doc/index.html

take place. A very simple example is that in which an object O might have to undergo processing 1, 2, 3 and 4, in that order. The graph $\overrightarrow{P}$ would then be a simple path where the arcs model the ordering (Fig. 1).

Fig. 1. A simple PFD

One of the remarkable problems is that of taking into account geometrical constraints. After other aspects have been considered, finally the designed system has to be implemented in some building. The PFD is to be transformed into a *Material Flow Diagram* (MFD) in which all aspects have been incorporated, but that then should fit into the building, i.e. satisfy the geometrical constraints. This step seems to cause considerable problems in practice and redesigning has to take place.

Smart Synthesis Tool (SST) aims to be a computer-based tool that should help engineers in designing solutions of high quality and in short time. Though the development of our ideas remains quite general, the SST can benefit from it, and it will be up to the software developer, to which extent our approach will best fit into the SST.
In this paper we discuss an approach that starts with the geometrical aspect. If a processing system is to be located in different parts of the building, possibly all in the same part, when e.g. only one big hall is disposable, then a natural first modeling of the building is by representing all available spaces by vertices and all possible direct connections between spaces, by doors or openings in walls, by edges between the vertices. In order to take into account that objects are delivered at an entrance and after processing are collected at an exit, one might model these two areas, or possibly more areas outside the building, by vertices too. In Fig. 2 a simple building is modeled by a *Geometrical Constraint Graph* (GCG).

The GCG has 8 vertices. The dotted edges connect neighbouring locations. Vertices 0 and 7 model entrance area, respectively exit area for the objects that are to be processed.
Our problem now is to embed the PFD of Fig. 1 in the GCG of Fig. 2. Any way to do this gives a potential solution to the design problem in a very abstract form yet. The vertices 1, 2, 3 and 4 are to be mapped on vertices of the set a, b, c, d, e, f, where we have chosen letters as labels to avoid confusion. As it may be that some processes take place at the same location, possibly all if there is only one location a, we assume loops on the vertices representing the locations. Both edges and loops can be represented by arcs, two in opposite directions in

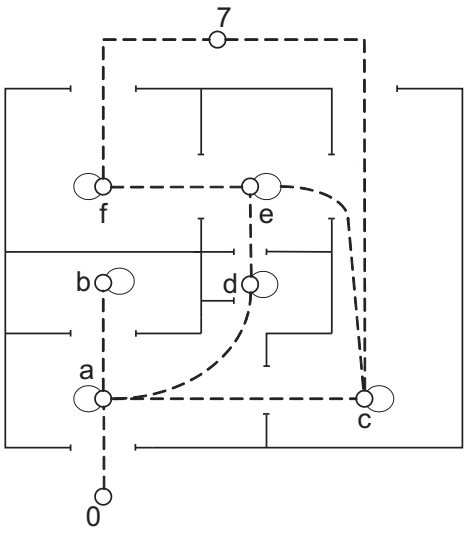

Fig. 2. A simple GCG

case of the edges. For the time being we assume that the whole building is available, i.e. no location is used for another purpose.

2 Mapping a *PFD* on a *GCG*

We will start with a pathological case.
The location b may be chosen as image of all four processes 1, 2, 3 and 4. The arcs are mapped on the loop on vertex b (Fig. 3).

This is a design, and about the least promising one. Much more likely to be satisfactory are mappings like

$1 \to a, \quad 2 \to d, \quad 3 \to e, \quad 4 \to f \quad or$
$1 \to a, \quad 2 \to c, \quad 3 \to e, \quad 4 \to f \quad or$
$1 \to a, \quad 2 \to c, \quad 3 \to c, \quad 4 \to c.$

In the latter case, the processes 2, 3 and 4 all take place at the large location c. It is clear that an SST can be produced, that gives all embeddings of the PFD on the GCG. Giving a designer the possibility to fix certain mappings, the SST will produce a smaller number of possible solutions. He/she will probably fix the mapping $1 \to a$. In case location c has to be left out of consideration, because of use in another way, probably $4 \to f$ will be fixed as well.

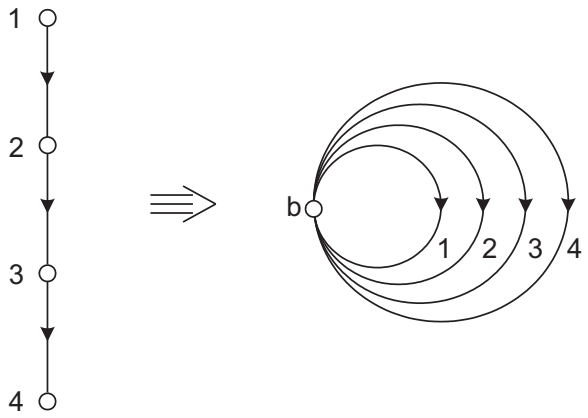

Fig. 3. Processes 1, 2, 3 and 4 taking place at location b

Now the number of possible designs has diminished considerably. We redraw the building, focusing on the inside, in Fig. 4.

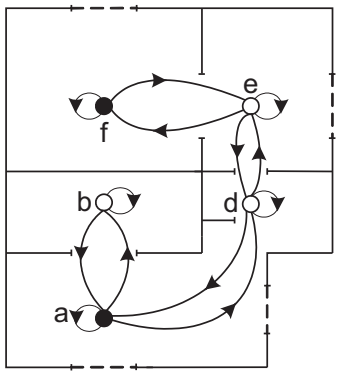

Fig. 4. Reduced GCG

Both vertices a and f have been drawn black as they are assumed to be the images of mapping vertex 1 and vertex 4 respectively. Let us now consider the possible abstract designs left. We will list them as sequences of locations, beginning with a and ending with f, only the images of processes 2 and 3 have to be chosen. The sequences are:

$a \to a \to a \to f \quad a \to b \to a \to f \quad a \to d \to a \to f$
$a \to a \to b \to f \quad a \to b \to b \to f \quad a \to d \to b \to f$
$a \to a \to d \to f \quad a \to b \to d \to f \quad a \to d \to d \to f$
$a \to a \to e \to f \quad a \to b \to e \to f \quad a \to d \to e \to f$
$a \to a \to f \to f \quad a \to b \to f \to f \quad a \to d \to f \to f$

$a \to e \to a \to f \quad a \to f \to a \to f$
$a \to e \to b \to f \quad a \to f \to b \to f$
$a \to e \to d \to f \quad a \to f \to d \to f$
$a \to e \to e \to f \quad a \to f \to e \to f$
$a \to e \to f \to f \quad a \to f \to f \to f.$

Several remarks have to be made now.

First, we have to make precise how the embedding of the PFD into the GCG is to be seen. The arcs can be interpreted as transportation channels. But if, as in $a \to a \to e \to f$, process 2 is to take place at location a, as was process 1, and process 3 is to take place at location e, the transportation of the object from a to e should be along a path from a to e, e.g. the path $a \to d \to e$. The embedding of the arc $2 \to 3$ of the PFD into the GCG is therefore not a mapping on an arc of the GCG, but on a path in which the vertex d occurs and the PFD from 1 to 4 is mapped on the path $a \to a \to d \to e \to f$ of the GCG. So a path of length 3 is mapped on a path of length 4. The image is, however, *homeomorphic* to the path that was mapped.

Second, several of the abstract designs would probably not be accepted by a human, like for example $a \to e \to a \to f$, unless there are stringent reasons for it. It might be so that process 2 can only take place at location e and process 3 only at location a. The reason it seems an unlikely design is that next to space, also time plays an important role. Transportation of the object from a to e, back to a and then to f will take more time than from a to a to e to f.
Yet we have now hit upon the next problem in deciding on the abstract designs to be considered further. Let us consider $a \to d \to d \to f$. This design lets processes 2 and 3 take place at location d, the smallest location in the building. Now processes may be performed by humans or by machines. In both cases enough space is needed. A machine will put a constraint on the space. Suppose processes 2 and 3 demand two machines, taking s_2 and s_3 as space, say in square meters. If the location d has space s_d and $s_d < s_2 + s_3$, then the processes cannot take place in d. In case neither process 2 nor process 3 can take place in d, not enough space for even the simplest configuration of machines, a single machine, the location d cannot be used as an image anymore. 9 out of 25 abstract designs are then removed from the list of possible designs. This simple space constraint may, for example, exclude that two processes take place at the same location. This may also be due to a decision of the designer who uses the SST. The se-

rious effect of these geometrical constraints becomes clear by considering the remaining possibilities for abstract designs.

$a \to b \to e \to f$ and $a \to e \to b \to f$ are the only two abstract designs still at the disposal of the designer! If now the time needed for transportation is considered, it is clear that the only abstract design that should be chosen is $a \to b \to e \to f$.

This simple example shows that an SST that incorporates these ideas can be of great help in reducing design possibilities. The programming will probably be quite straightforward.

3 Incorporating other constraints

Figure 5 gives the outcome of the first step in the design process, with help of the SST.

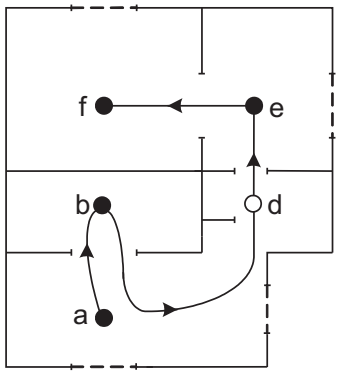

Fig. 5. Abstract design taking into account geometrical constraints and transportation time considerations

The design says: start with process 1 at location a, transport to b, let process 2 be carried out there, transport to e, via a and d, let process 3 be carried out there, transport to f and carry out process 4. Now we have to see how this embedded PFD can be turned into an MFD.
Suppose, again in our example, N objects have to be processed in a certain time period. This puts requirements both on the processes 1, 2, 3 and 4 and on the paths that, in an abstract sense, represent the transportation. The means of transportation should have enough capacity, as should have the humans or machines carrying out the processes. In both cases the second step in the design comes forward. The available ways for transportation and processing may lead

to *multiplication*, again in an abstract way.

4 More complex processing

Let us imagine people carrying out both transportation and processing. If one person can process n_i objects in the prescribed time period, at least $\left\lceil \frac{N}{n_i} \right\rceil$ persons are needed for process i. In case of machines, a minimum number of machines is needed as well. We now meet the geometrical constraint again. The locations may not have enough space for the machines needed. This way an upper bound on N is found, unless there is the possibility to choose from machines with different capacities and, probably, different prices!

Consider Fig. 5 from the previous section. We decide to find the *best* (not referring to any specific performance as time or cost, we still keep it general) suitable number of machines able to perform the process number 3. Assuming that the space required for such a machine is known, and considering the area s_e known, it may be that we decide for a number of three machines necessary to be used. Now the challenge is to properly locate these three machines into the given area s_e. The problem is involving not only the geometry (i.e. where exactly to locate the machines), but also taking into account the access to the entry in the space s_e (from s_a) and also the exit to the next area s_f (with respect to the direction indicated on the depicted graph). At this point it is important to place the machines m_1, m_2 and m_3 such that the process 3 can be undertaken in an optimal time and that areas $s_{e,1}, s_{e,2}$ and $s_{e,3}$, corresponding to each of the machines do not impede each other.

Two problems can be distinguished and formulated with respect to this complex processing issue. On one hand, determining how many processors/machines are necessary, depending on capacity requirements, is a well-defined problem. Hence the number of processors needed for the process 3 in location e can be found.

Let there be m_t machines available, of type t, $t = 1, \ldots, T$. A machine of type t may be able to process c_t objects per time unit, and demand space s_t. Let p_t be the price of this type of machine. If s_e is the available space at location e, $a_t = \left\lfloor \frac{s_e}{s_t} \right\rfloor$ is the number of machines of type t that can be used. The capacity of these a_t machines is $a_t \cdot c_t$, their price is $a_t \cdot p_t$.
For any combination of different types of available machines, both capacity and price are easily calculated as

$$C = \sum_{t=1}^{T} a_t \cdot c_t, \text{ and } P = \sum_{t=1}^{T} a_t \cdot p_t.$$

The numbers a_t determine a space demand $a_t \cdot s_t$, so that we have the constraint

$$\sum_{t=1}^{T} a_t \cdot s_t < s_e$$

for the combinations of machines to be used at location e. The availability can be expressed by

$$a_t \leq m_t,\ t = 1, \ldots, T.$$

If the desired capacity c_{min} is known, our problem may be to determine a combination for which $C \geq c_{min}$ and P is minimized. If the price is not a point, a natural question is for which combination c_{max} is obtained.

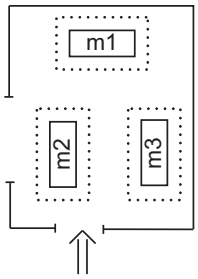

Fig. 6. Placing three machines

On the other hand, the way how these processors can be placed in the given area is a challenging point, so that the best use of the available space is made, whereas the processing does not suffer any delay.
In Fig. 6 one type of machine is placed. The other possible solutions of combinations of machines, satisfying the requirements, might claim certain space per machine, leading to a geometrical *placing problem*.
Not only the shapes of the required space may be different, also the transportation to and from the machines has to be taken into account. The SST will have to offer the combinations that satisfy the constraints. This can probably be programmed in a rather straightforward way.
The SST should also furnish possible placements. However, even if transportation is not taken into account, that geometrical problem seems of considerable difficulty. Most probably, the human designer should consider the offered combinations and solve the placing problem, getting at most some help from the computer in the form of a drawing in which machines can be shifted in location.

In case there is no room for the required processors at location e, the multiplication of processors leads to another problem. Some of the processors have to be

located elsewhere. The design can then be restarted by multiplying the process considered by a factor 2, 3, etc. and replace the vertex corresponding to the process by 2, 3, etc. vertices with the same labels in the PFD and connected to the vertices the single vertex was connected to.

5 Redundancy and vulnerability

Multiplication of processing machines or human processors has a very important consequence for the designed system. We will illustrate the definitions that we will give by an extremely well-known and simple example, the cash registers in a supermarket.
If we assume an entrance location 0 and an exit location 2, and only one location a for process 1, the system has a GCG as in Fig. 7,

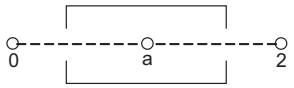

Fig. 7. Process 1 taking place at location a

and the PFD consists of the single vertex 1. Vertex 1 of the PFD can only be mapped on vertex a, that we now can investigate on the number of machines, the cash registers, that are to be used. We assume that there are no geometrical constraints. The number of machines depends on the number of objects to be handled, customers, and on the number of cashiers available. We assume that there are always enough people to handle the machines that are present.
The minimum capacity c_{min} of the cash registers depends on the flow of customers, the average time needed per customer and on the policy of the supermarket. Usually a new cash register is opened when there are more than a certain number, say three, customers who are waiting at the open cash registers. The knowledge about the flow in busy times determines c_{min}.
Suppose c_{min} asks for eight cash registers and ten cash registers are present, with c_{max} as total capacity. We can now discuss several things more precisely. First, there is the change in the PFD that now consists of ten processors 1. Graph theoretically it is $\overline{\mathcal{K}}_{10}$, the complement of the complete graph on ten vertices, or just a set of ten independent vertices. The GCG, after this multiplication of processors by a factor 10, is given in Fig. 8.

Secondly, we give a definition of redundancy, or "superfluousness", or "overcapacity".

Definition 1 The *redundancy* of an object processing system is $c_{max} - c_{min}$.

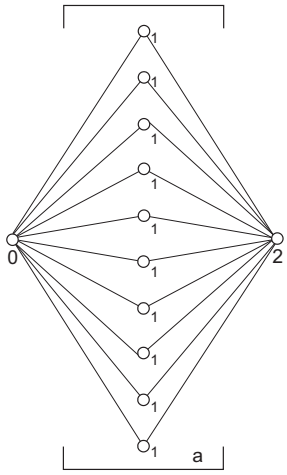

Fig. 8. *GCG* after multiplication. All processors at location a.

In our example, the redundancy of the design is the capacity of 10 - 8 = 2 machines. During not so busy shopping times more machines may stand idle. If only three cash registers are open, there are seven machines not used. The concept of redundancy should, however, only be used with respect to the value c_{min} of the design.

The reason to have redundancy, overcapacity, at all is the fact that machines may fail to work. With two cash registers not working, c_{min} can still be achieved.
The concept of redundancy should clearly be distinguished from that of *vulnerability*. The failure of a processor is lethal for the system if there is only one processor. The flow from 0 to 2 is completely interrupted in our example. Distributing the flow over more processors reduces the vulnerability. For that reason, any combinations of processors satisfying the constraints may be compared on the number of processors in the corresponding designs.
The mathematical definition of vulnerability can be based on that of connectivity. We consider a connected graph with two distinguished vertices, a *source* s and a *sink* t, in our example the entrance and exit vertices 0 and 2.

Definition 2 The *vertex connectivity* of a connected graph with source s and sink t is the minimum number of vertices that have to be deleted to disconnect s from t.

In Fig. 8 the vertex connectivity is 10. For a single cash register it would be 1.

Definition 3 The *vertex vulnerability* of a connected graph is the inverse of its vertex connectivity.

In Fig. 8 the vertex vulnerability is $\frac{1}{10}$. For a single cash register it would be 1, the maximum value for a connected graph.
There are analogous definitions for *edge (arc) connectivity* and *edge (arc) vulnerability*.
These will play a role in the next section, where we consider multiplication of transportation machines for the same reason. As these too have capacities, the Definition 1 of redundancy applies to them as well.

6 More complex transportation

We come back to our first example. Process 1 in a, 2 in b, 3 in e and 4 in f can be designed separately with respect to the capacity considerations. A similar situation holds for the *transportation* designs. Each abstract path will have to be transformed into a real life transportation system. To think in terms of humans again; the transportation from b to e will demand more persons than the transportation from a to b or from e to f. The problem is to determine the number of people needed. The designs consist of number of people carrying the objects and the routes they should optimally take. These routes become of great importance once automatic transportation machines are used and there must be more than one of them. Yet, here too in first instance, in our simple example, the design problem can be solved for each path separately.

Once we have decided how many processors each of the given spaces claims, the main problem consists in finding out all the existing possibilities to make the connections between different 'rooms' of the whole building (see Fig. 9).

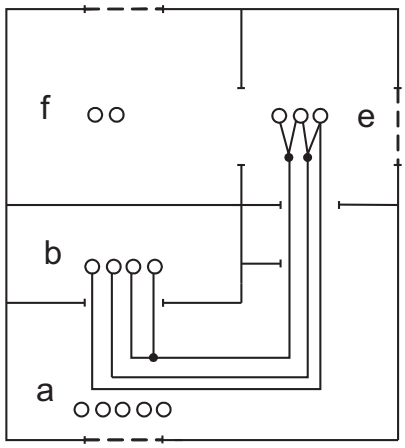

Fig. 9. Transportation alternative between 'rooms', involving fusion/splitting

When trying to analyse how the generation of transportation layouts can be done, we can say that, basically, the problem is that the SST should assist in generating all possible connections from the processors of one phase (corresponding to n_1 processors) to the processors of a next phase (corresponding to n_2 processors).
In Fig. 10 we consider two consecutive processor types, that are connected by an arc in the original PFD, and the resulting situation after multiplication of the processors by n_1, respectively n_2. The problem is how the arc, that describes the transportation in an abstract way, is to be replaced to obtain a design of the transportation. In the figure, the small circles represent "splitting points", respectively "points of fusion". In a way, this is a first design, the n_1 processors produce objects that are gathered at a point of fusion, transported by one transportation machine and then distributed over the n_2 processors at a splitting point.

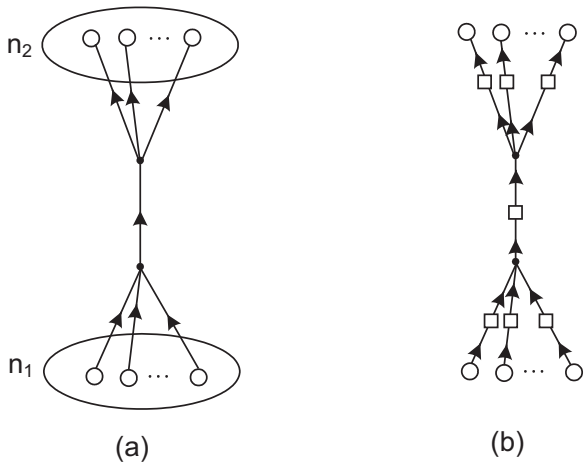

Fig. 10. Basic transportation between two groups of processors

Vulnerability considerations and redundancy considerations lead to the problem of transforming the basic transportation design into designs of lower vulnerability and, possibly, of higher redundancy. We want to find a set of types of transformation, that enables us to generate all possible layouts. We will restrict ourselves to planar layouts, that is layouts in which transportation arcs do not cross. We will also assume that the location poses no geometrical constraints. This assumption is also made if, along with the actual system design, also the building still has to be designed.

For our presentation it is advantageous to represent the directed graph in "total" graph form, in which also arcs are represented by vertices, for which we shall use squares. These squares represent the transportation machines. They are co–nnected according to the graph structure, where splitting and fusion points are acting, like the processors, as "normal" vertices, see Fig. 10(b).
The first type of transformation, T_1, describes multiplication of one transportation machine. The transportation is distributed over a certain number of parallel machines. In Fig. 11 this is illustrated, as well as the transition to the total graph form.

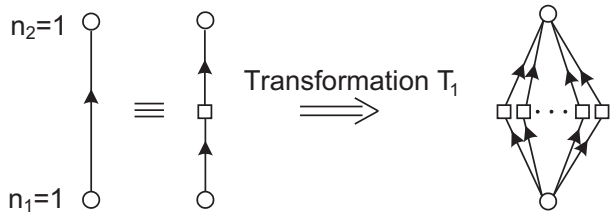

Fig. 11. Multiplication of one transportation machine

The second type of transformation, T_2, describes the multiplication of splitting points and fusion points. Without this transformation, the vertex vulnerability would be high in spite of the multiplication of the transportation machines, as the splitting or fusion may fail.
In a general way the two processor groups are partitioned into subgroups, so n_1 processors are partitioned into subgroups of $n_{1,1}, n_{1,2}, \ldots, n_{1,k}$ processors and n_2 processors are partitioned into subgroups of $n_{2,1}, n_{2,2}, \ldots, n_{2,l}$ processors. The basic transportation design is transformed into a design with k splitting points and l fusion points. These are now first to be connected to the processors and then to each other, see Fig. 12.

The planarity constraint enforces placing processors in subgroups together, and also puts constraints on the ways the main transportation between fusion points and splitting points can be designed. In Fig. 12, connecting the subgroup of $n_{1,2}$ processors to the subgroup of $n_{2,1}$ is not possible, due to the presence of the connection between the $n_{1,1}$ processors with the $n_{2,2}$ processors.
Several interesting problems can be posed concerning the layout problem for transportation machines. First, there is, of course, the problem of satisfying capacity and price constraints on transportation machines, similar to that for the processors.
Secondly, there is a problem concerning the partitioning of the two groups of processors, with respect to the capacities, that should match. This problem is not present if there is only one splitting point and the splitting according to the

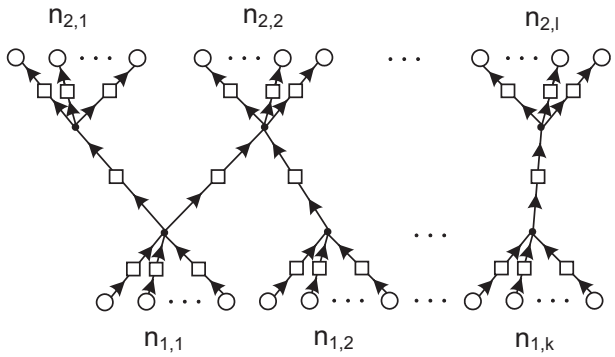

Fig. 12. Transformation T_2. Multiplication of splitting and fusion points.

capacities does not pose a problem.

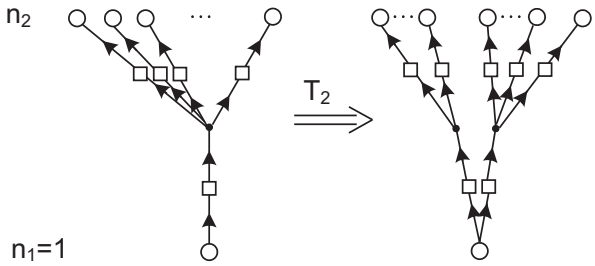

Fig. 13. Multiplication of a splitting point.

Figure 13 depicts a situation where having to transport from a single machine to a number n_2 of machines, a possible alternative, from many existing, appears. In order to reduce vulnerability, two paths are splitting from the starting point. The n_2 machines are partitioned into two subgroups. The splitting point is multiplied by 2. Now the capacities of the two transportation machines should match that of the two subgroups.
Thirdly, interesting problems arise if transportation can take place on more than one level. The layout need then not have to be a planar graph.

We close our discussion by showing how the layout of Fig. 9 can be achieved by applying our two transformations.

Figure 14 gives the successive transformations.

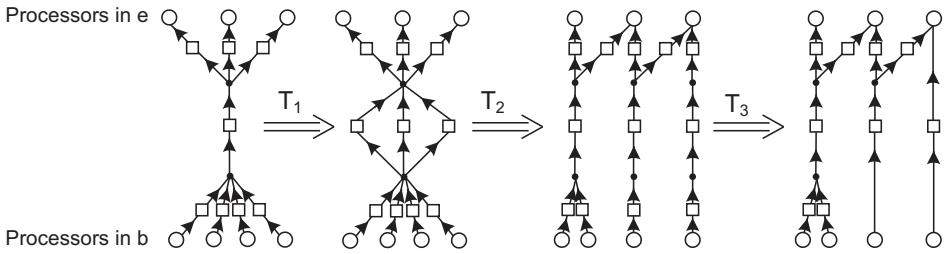

Fig. 14. Generation of a transportation layout.

Note that in T_2 new transportation machines are generated as well to effectuate the connection between processors and splitting and fusion points. Also note that a third transformation T_3 is involved in which two, respectively three transportation machines are replaced by just one transportation machine, see Fig. 15.

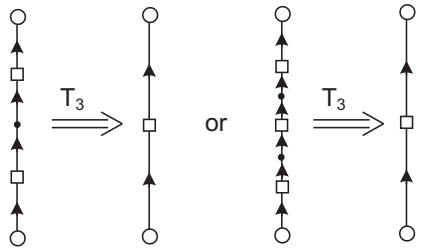

Fig. 15. The transformation T_3

Finally note that there may not exist any machines with fusion or splitting abilities. The transformations T_2 and T_3 will then have to be applied until only direct arcs between the processors of the two phases remain.
The SST should be able to furnish all possible layouts that can be generated by the three transformations.

7 Summary

In this paper we have presented a graph theory-based approach to designing object processing systems. Object processing systems are omnipresent in today's world. In particular in the area of logistics[2] one can find myriads of examples of systems that have the aim to process objects in an effective and efficient way. We have considered the simplest Process Flow Diagram only, namely the simple path. However, very often objects have to be sorted and the flow is split into two or more flows.
Let us think of the selection of mail and its distribution to different destinations. First, the mail is collected by a number of operators from different points of, say, a city, to a central office, and from there, afterwards, a selection is done, according to different cities, countries and so on. This is a well-known example of a system with sorting.
An interesting and complex example is offered by baggage handling systems at airports. The handling of passenger baggages includes passing several points in the baggage handling system, like e.g. check-in and screening points. Sorting points come in as soon the flow of bags, the objects, is split. This may be a consequence of screening, manual coding, etc.

We reduced the generation of possible designs for an object processing system in this paper by first solving a graph embedding problem. We proposed three types of graph transformations which are easing considerably the way the behaviour of object processing systems can be characterized. The next step would be to make sure whether for some specific case studies of object processing systems, the three found transformations suffice in obtaining designs as solutions, and if not, finding out in which way the present approach can be extended, so that it would offer better results.

In a second paper we will apply these graph transformations to existing so-called Material Flow Diagrams, and thereby provide a first step to the final design of such systems.
On the long term, our present approach should lead to an SST that offers a formal way to translate a schematic PFD into a better representation, which will serve as intermediary step towards the design implementation itself.
As the ideas brought forward in this paper are not based on existing theories, the bibliography contains only some general background information.

To summarise this paper, we give a description of how an SST might function on the basis of the ideas developed here. The SST may perform the following tasks:

[2] See `http://www.websters-online-dictionary.org/definition/logistics`

TASK/PROBLEM	**SUPPORT**
Choosing the PFD	The SST offers a survey of possible PFD's from which the designer may choose.
Mapping the PFD on the GCG	The SST calculates all possible mappings of the chosen PFD on the GCG.
Multiplication of the processors	The SST uses a survey of the available processors, calculates the desired number of processors and checks whether the chosen multiplication is realisable at the location given by the mapping of the PFD on the GCG.
Placing the processors	The SST offers the designer help in placing the processor at the intended location.
Multiplication of the transporters	The SST can be given a survey of the available transportation systems, then calculates the desired number of transporters and checks whether the chosen multiplication of transporters is achievable.
Generation of layouts	The SST offers a survey of possible layouts of the transportation system, indicating whether the layouts can be realised in a planar way or necessarily involve crossings.
Compactification	The SST offers alternatives for layouts that require less space, due to multi-level design, so that crossings in the designed transportation system can be admitted.

Table 1. Summary of potential support of an SST

In the forthcoming paper we will in particular focus on the presence of sorting processors. These lead to PFD's that do not have the simple form of a path. We will also show how two actual designs of object processing systems can be generated by the means developed in this paper.

Bibliography

[BE] Bystrom, M., and Eisenstein, B., (2005). *Practical engineering design.* Boca Raton, FL: CRC Press.

[BM] Bondy, J.A., and Murthy, U.S.R., (1976). *Graph Theory with Applications.* London: Macmillan.

[Die] Diestel, R., (2005). *Graph Theory.* Berlin: Springer.

[Dov] Dov, D., (2002). *Object-process methodology: a holistic system paradigm.* Berlin: Springer.

[GP] Goldschmidt, G., and Porter, W., (2004). *Design representation.* London; New York: Springer.

[Gue] Guelzo, C.M., (1986). *Introduction to logistic management.* Englewood Cliffs, N.J.: Prentice-Hall.

[HKS] Hruby, P., Kiehn, J., and Scheller, C.V., (2006). *Model-driven design using business patterns.* Berlin: Springer.

[Sch] Schoon, J.G., (1996). *Transportation systems and service policy - a project-based introduction.* New York: Chapman&Hall.

[Ull] Ullman, D.G., *Towards the ideal mechanical engineering design support system.*, Research in Engineering Design, Vol. 13, 2002, pp. 55–64.