INFORMATICA, 1999, Vol. 10, No. 1, 89–108

© 1999 *Institute of Mathematics and Informatics, Vilnius*

Automatic Configuration of Structured Graphical Documents: A Case Study in the Domain of Electroplating Lines

Kristina LAPIN
Vilnius University, Department of Computer Science
Naugarduko 24, 2600 Vilnius, Lithuania
e-mail: Kristina.Lapin@maf.vu.lt

Received: January 1999

Abstract. In this paper we present a method for preparation of technical drawings in the domain of electroplating lines. We treat the method as the combination of two tasks: configuration and document preparation. The proposed method combines the basic concepts of these domains: generic coding introduced in document preparation task and component oriented approach used in the configuration task. We show how both concepts are implemented in our configurator SyntheCAD.

Key words: knowledge-based configuration, document preparation, silicon compilation.

1. Introduction

Document preparation involves two tasks: defining the content and structure of a document, and generating the document from specification of its appearance. The first part is usually called editing while the second, is known as formatting. Our aim is to characterize both aspects of graphical document processing.

In order to discuss the document preparation, we use an object model of documents, described in (Furuta *et al.*, 1992). *A document* we understand as an object composed of a hierarchy of more primitive objects. Each object is an *instance* of a *class* that defines the possible constituents and representations of the instances. Some typical document classes are books, articles, business letters, reports, papers for a journal or a conference, programs in a given language, etc. Common lower-level classes include such document components as sections, paragraphs, headings, footnotes, tables, equations, figures, etc. Objects futher are classified as either *abstract* or *concrete*. To each abstract object, there corrensponds one or more concrete objects. An abstract object is denoted by an identifier and the class to which the object belongs. The term *logical object* is used as an informal synonym for an abstract object. Concrete objects are defined over one or more two-dimensional page spaces and represent the possible formatted images of an abstract object.

A lot of tools are developed for processing of textual documents. They maintain the structure of logical objects and are not suitable for graphical documents. Graphical tools

heavily support the structure of documents. Such documents are difficult to modify. We try to solve these shortcomings in our tool SyntheCAD.

In the following section the problem domain of electroplating lines is outlined. We give a hierarchy of drawings that we have to prepare. In the third section we deal with an overview of methods that are integrated in our tool. The most detailed is the fourth section which reports the conceptual model of SyntheCAD. The emphasis is on SyntheCAD specification languages, functions performed by the formatter, and the integration of formatters with other document processing tasks. Finally, some conclusions are drawn.

2. Problem Domain

In this section we describe the domain of electroplating lines.

2.1. *An Electroplating Line*

An electroplating line is a conveyor-based array of *processing units* (Fig. 1). One or two transporters move above the electroplating line.

The transporters put processed elements into a processing unit. After a certain time segment the transporter takes the element and put into the next processing unit. The processed element goes through all the processing units in the electroplating line.

All processing units are monitored by controllers. For example, a heater is turned on, when the temperature of the solution in a bath is smaller than a certain value. The controller turns the heater off, when the solution temperature reaches a required level.

An electroplating line has global attributes: type, width, number of transporters, etc. It is configured from a fixed given set of *typical* components. Several components can be of the same type. The component types in our domain are: processing units (baths, loaders, stores, dryers), transformers, pumps, etc.

A bath is the major processing unit and has the following attributes:

- function (electromechanical, washing, etc.),
- characteristics (dimensions, material, etc.),

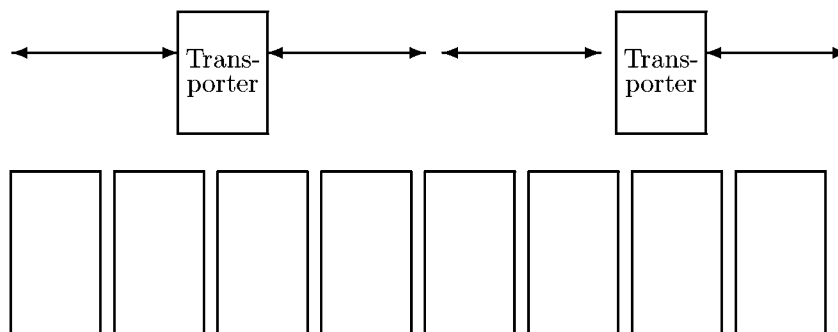

Fig. 1. Electroplating line.

- optionally attached equipment (temperature sensors, filtration units, agitation device, pump, etc.),
- associated devices (subunits) (rectifier, reservoir, safety level sensors, heaters, cooler, steam/water electrovalves, etc.).

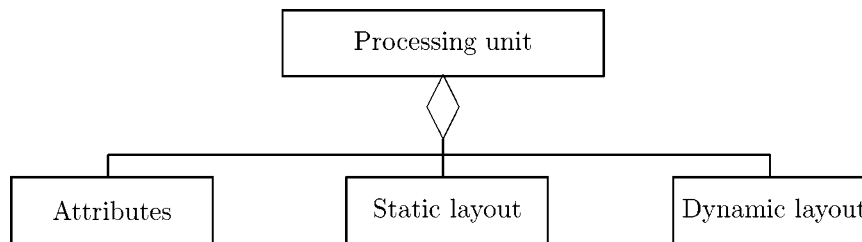

Fig. 2. Information associated with processing units.

The knowledge associated with every processing unit is of three kinds (Fig. 2). A function, characteristics and a list of optional equipment are the attributes of a processing unit. The technical specification of an electroplating line specifies the types of all processing units. The subunits are represented as child-nodes to bathes. A component is shown on the drawings as a two-dimensional image.

The domain of electroplating lines developes constantly. New equipment and new modifications of the processing units are developed. The graphical representation of the existing equipment can also be changed. This change causes changing connections algorithms. Therefore, the most frequent activities during design of technical drawings are as follows:

- introducing new modifications of units,
- modifying the images of units,
- changing the algorithms of connections.

New versions of CAD tools are developed. Therefore, the user should have a possibility easy to adapt an output for new needs.

2.2. *Domain Decomposition*

The natural decomposition is observed in the domain of electroplating lines. Fig. 3 shows the structure of technical documentation. From the viewpoint of document preparation, the technical drawings are structured documents. A set of technical documents is partitioned into mechanical drawings and electrical ones. A set of electrical drawings consists of the diagram of connections and the device layout. Mechanical drawings consists of the assembly drawings, the piping plan, the framework and the fume ventilation plan.

Each subset is divided into pages. The components (processing units and etc.) are shown in the context of a concrete diagram. For example, in the diagram of connections the electrical connections of the baths are shown. In the piping diagram the piping of baths is shown.

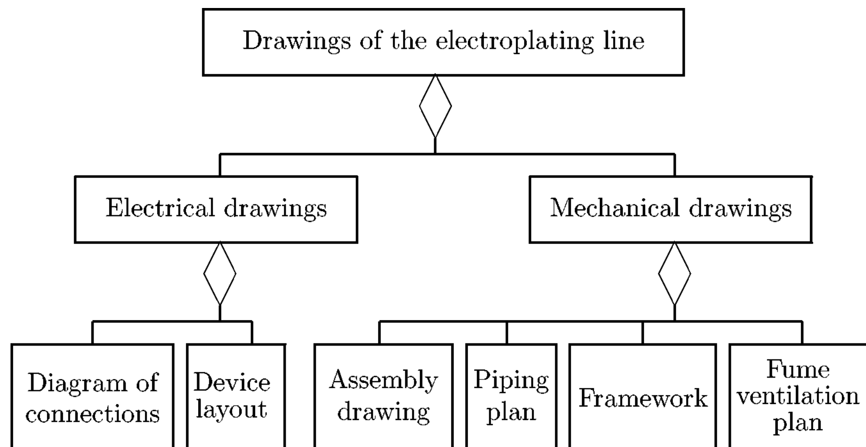

Fig. 3. Structure of a drawing set of the electroplating line.

3. Overview of Used Methods

The proposed approach to synthesis of technical documentation is based on the concepts of:

1. generic coding (separation of logical and layout aspects of the problem domain concepts);
2. integration of variuos data representation methods:

- arrangement language for a SyntheCAD document in order to express a hierarchy components with their attributes;
- high-level descriptive GCL language for representation of a knowledge of technical specification and expert users. Knowledge is represented as in the Generative Constraint Satisfaction Problem (Stumptner, 1997). Variables can be generated as needed (to represent the adding of components to a configuration). Introduced components are subject to a set of constraints depending on the component type;
- a general purpose CAD tool for static layouts and the visualisation of drawings.

3.1. *Generic Coding*

The concept of generic coding gained prominence in the 1970s and was implemented in Standard Generalised Markup Language (ISO-Standart 8879, 1986). Earlier, electronic manuscripts contained control codes that caused the document to be formatted in a particular way. That was 'specific coding'. In contrast, generic coding, which began in the 1960s, uses descriptive tags ('section' rather than '14 point Palatino'). Central to the concept of generic coding is the separation of the *information content* of documents from *the format* or *the appearance* of the content. An information content has a structure, then we talk about logical structure of the document (Wilhelm and Heckman, 1996). The structure

of the appearance of a document we call *layout structure*. Eickel (1990) deals with the logical and layout structures of documents. These concepts make background for Document Style Semantics and Specification Language. The aim of separating the logical and layout structures is to achieve the flexibility in the preparation of structured document. By flexibility we mean multiple specification that may be applied to a given document to yield various presentations of the same data.

Document processing consists of executing various operations to define, manipulate and view abstract and concrete objects. For this purpose, we distinguish between *ordered* and *unordered* objects.

In the drawings of electroplating lines we have ordered objects – processing units. The one-dimentional order is defined by the user. The hierarchy of units forms a logical structure of the document. The logical structure of an electroplating line consists of:

- the attributes of the whole line;
- processing units with their attributes and associated devices (subunits) with their attributes.

The layout structure of an electroplating line is as follows:

- A drawing is divided into pages. Each page is partitioned into the areas for:
  - global properties (attributes),
  - controllers with cables (in the case of electrical drawings),
  - transporters,
  - associated devices,
  - processing units. This in turn consists of smaller parts :
    - static image,
    - dynamic layout (cables, pipes).

We treat the document preparation as a configuration task.

3.2. *Configuration of Technical Drawings*

Configuration deals with assembling of complex systems from a set of simpler components. A configuration system is expected to produce a new artefact. In the domain of electroplating lines a new instance of an electroplating line is created from an individual order with unique customer requirements. The knowledge is taken directly from the technical specification of the components (processing units) and is also derived from long term experience with actual configurations. From the configuration point of view we show the place of our approach in the classification (see Fig. 4) proposed in (Stumptner, 1997).

The area of configuration systems is divided into two major subgroups: representation oriented and task oriented. We deal with representation oriented approach. The central issue in this approach is finding the correct representation for expressing the structure and properties of the problem domain. In first-generation expert systems knowledge representation was based on production rules in the form IF *condition* THEN *action*. In structure based configuration *the key component* approach (Mittal and Frayman, 1989) was

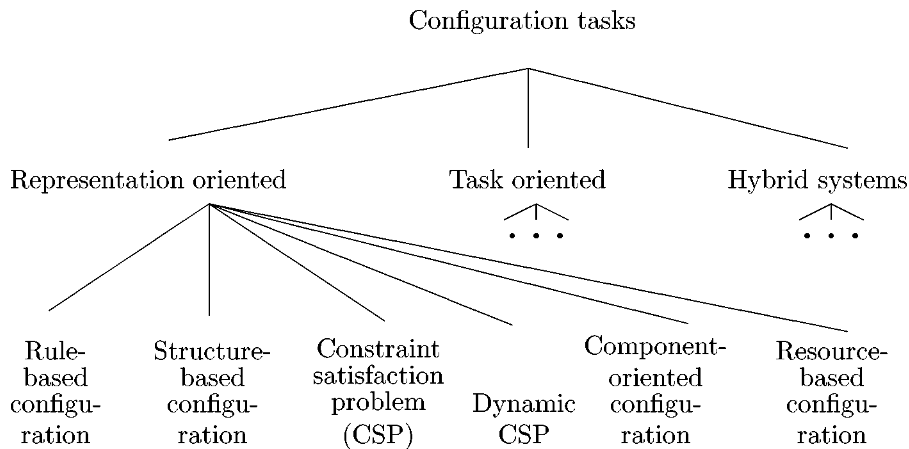

Fig. 4. Classification of configuration tasks according to Stumptner (1997).

defined. In this approach components are complex objects which can be connected in different ways. The key components are those components that are central to the architecture of the system, and their choice is a basic decision in modelling a particular artefact. A constraint satisfaction problem (CSP) is considered to be a formalism for describing application specific configuration knowledge. This formalism is used in expressing expert knowledge, that have been described as a result of actual experience. Dynamic CSP extends the restriction of CSP that postulates the existence of a fixed, enumerated set of variables. The main goal of a configurator in the case of resource-based configuration is to select a correct set of components, based on the functionality they provide. The number of components in this case is not predefined. Component-oriented configuration attempts to combine the advantages of structure-oriented and resource-oriented configuration. Instead of a fixed kit of individual components, we have the set of component *types*, with each type determining the structure and constraints of its instances. So, an electroplating line is configured from a fixed given set of component types. A component we understand as a processing unit with a set of attributes which describe the behaviour of parts to be assembled.

During the component-oriented configuration are performed three basic actions:

1. creating components;
2. choosing type for a component;
3. connecting two components.

According to the component oriented approach our task is to formalise the knowledge of an expert user in the domain of electroplating lines.

3.3. *Silicon Compilation*

The term of *silicon compilation* was introduced by Dave Johannsen at the California Insitute of Technology, where he used it to describe the concept of assembling parameterized pieces of layout. In (Gajski and Thomas, 1988): “Silicon compilers are the programs

that generate layout data from some higher-level description. This higher-level description can be a symbolic layout, a circuit schematic, a set of boolean expressions, a logic schematic, a behavioural description of a microarchitecture, an instructuin set, or just an algorithm for signal processing. Silicon compilers contain knowledge of how to synthesize higher-level constructs from basic components and how to arrange components and intersections on silicon. This knowledge is stored in the form of design rules that conform to constraints imposed by the technology used."

Silicon compilation helps resolving "hardware crisis": the great difficulty encountered in the generation new, custom integrated circuit microchips. The progress in single-chip microprocessors technology brought great increase in the complexity of the design process. The bottleneck involves the translation from the human intent, a behavioural description what the chip is supposed to do, into the chip "layout", a complete geometrical representation from which physical technologies can readely produce reliable, working chips. The silicon compilers were introduced to deal with this increased complexity.

There are two focal points of concern in a silicon compiler:

- the source language: the language in which the user specifies desired function, or behaviour, to be performed by the new integrated circuit chip;
- the target language: the capabilities of silicon, utilised in a very complex two-dimensional colour picture called a layout.

Traditional methodologies require a designer to build a structure and define its behaviour with basic components such as gates, and then use it hierarchically to build high-level structures. The silicon compilation method provides basic components which are fine-tuned to the use's specification at a higher level of abstraction, such as units, controllers, etc.

Silicon compiler systems come in two varieties: they may be complete systems with their own set of proprietary tools for complete integrated curcuit design or integrated into a standart Common Application Environment workstation with most of the support tools provided by the host workstation environment. The second approach offers a transition from existing design methods and the possibility of reusing parts already designed. In (Ayres, 1983) much more operations, such as process type are defined. Our domain does not require the process concept.

Silicon compilation takes advantages in three areas:

- usability,
- quality,
- profitability.

Silicon compilation improves design quality through "correct by construction" capability. The design errors introduced by designer on more abstract levels are easier to detect and correct. Instead if dealing with millions of polygones or transistors, the designer must deal with only two dozen microarchitectural blocks.

A principles of silicon compilation are implemented in SyntheCAD formatter language GCL (Section 4.2.2).

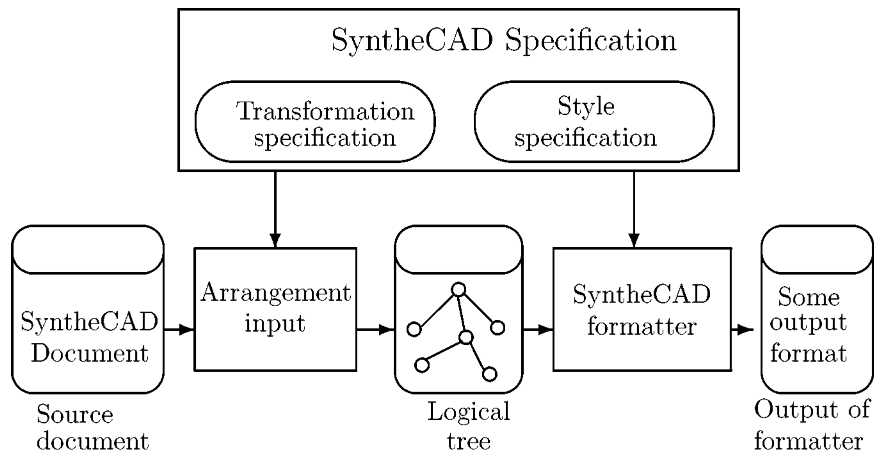

Fig. 5. The conceptual model of SyntheCAD.

4. Conceptual Model of SyntheCAD

The conceptual model of SyntheCAD (Fig. 5) has much in common with the conceptual model of DSSSL (ISO-Standart 10179, 1996). It has two distinct processes:

1. a transformation process, and
2. a formatting process.

Every SyntheCAD process is controlled by an appropriate SyntheCAD language. The transformation language controls the transformation process. Likewise, the style language controls aspects of formatting process.

A SyntheCAD document is specified by arrangement language (see Section 4.1). A logical tree is formed by the logical parser-tree, which is decorated with synthesized attributes. Parser-tree nodes represent components. The synthesised attributes represent properties of the components.

We deal with the formatting process in Section 4.2.1.

4.1. *The Arrangement Language*

For describing of both the document type definition (DTD) (Maler and Andaloussi, 1996) of a SyntheCAD document and a SyntheCAD document, a context-free language was developed. SyntheCAD document specifies a logical structure of the technical drawings. Comparing with a classical attribute grammar formalism our language allows to describe the attributed tree shorter and without superfluous redundancy.

In configuration area, a document type definition play the role of *Arrangement Template* (Čyras *et al.*, 1993). A SyntheCAD document with a concrete logical structure serves as an arrangement of an of electroplating line's instance. The mentioned context free language is called *arrangement language*. The syntax of the DTD of an electroplating line might be shown as a grammar:

```
SyntheCAD_document → head_component ( attributes* ) ⟦ component+⟧
component          → component_id  (attributes*) ⟦ component* ⟧
attributes         → positional_attribute* keyword_attribute*
positional_attribute → type_id attribute_id
keyword_attribute  → attribute_id <  value_set > =? default_value?
attribute_id       → string
component_id       → string
value_set          → value,* value
type_id            → int | real | bool | string
default_value      → int | real | bool | string
```

4.1.1. *DTD – the Arrangement Template*

The *Arrangement Template* represents a syntax of arrangement class. It describes artefact's model: the names and the "part of" hierarchy of nodes, field names, etc. An arrangement in Section 4.1.2 corresponds to the following arrangement template:

```
PLATING_LINE ( string type,                                  /* ID of line */
               width <1120, 1500 >,       /* Width: 1120 mm or 1500 mm */
               tranporters <1, 2, 3, 4>=2,      /* Number of transporters */
               direction <direct, reverse>= direct            /* Direction: */
             )                                /* right to left or left to right */
             ⟦ BATH      ( string type, int position, drum <yes, no> )
               ⟦ LEVEL_SENSOR
                 STEAM_VALVE
                 RESERVOIR
                 RECTIFIER ( string type )
               ⟧
               DRYER                          ( string type, int position )
               LOADER                         ( string type, int position )
               STORE                          ( string type, int position )
             ⟧
```

4.1.2. *An Example of a SyntheCAD Document*

A SyntheCAD document represents a domain-specific logical structure, termed *arrangement* (Brown, 1997). In arrangement specific relationships are established. Specific relationships, used in arrangement, will precisely locate one component with respect to another or with respect to some reference location.

The main information contained in the arrangement is a sequence of baths. The domain matches the general definition of a configuration task as well as two assumptions (Mittal and Frayman, 1989) "functional architecture" and "key component per function".

Arrangement Language terms originated during analysis of the problem domain. The purpose and properties of bath types and other components was analised. An example below illustrates a sample customer's order, i.e., an arrangement:

```
PLATING_LINE ( 443214.001_E4, 1120, transporters = 1, direction = reverse )
  ⟦             /* Type, position, drum */
    LOADER  (  009     , 1 )                          /* Load and unload */
    STORE   (  057     , 2 )                          /* Storing to wait */
    DRYER   (  001     , 3 )                          Drying with hot air */
    BATH    (  002     , 4       , yes )        /* Rinsing in hot water out*/
      ⟦ LEVEL_SENSOR STEAM_VALVE ⟧
    BATH    (  001     , 5       , yes )        /* Rinsing in cold water out*/
      ⟦ LEVEL_SENSOR ⟧
    BATH    (  006-01 , 6        , yes )             /* Chemical treatment */
    BATH    (  001     , 7       , yes )        /* Rinsing in cold water out*/
      ⟦ LEVEL_SENSOR ⟧
    BATH    (  006-01 , 8        , yes )             /* Chemical treatment */
    BATH    (  001     , 9       , yes )        /* Rinsing in cold water out*/
      ⟦ LEVEL_SENSOR ⟧
    BATH    (  006-01 , 10       , yes )             /* Chemical treatment */
    BATH    (  009-03 , 11       , yes )                  /* Electroplating */
      ⟦ RECTIFIER(TBPI-1600/12) ⟧
    BATH    (  009-01 , 12       , yes )                  /* Electroplating */
      ⟦ RECTIFIER(TBPI-1600/12) ⟧
    BATH    (  001     , 13      , yes )        /* Rinsing in cold water out*/
      ⟦ LEVEL_SENSOR ⟧
    BATH    (  006-01 , 14       , yes )             /* Chemical treatment */
    BATH    (  001     , 15      , yes )        /* Rinsing in cold water out*/
      ⟦ LEVEL_SENSOR ⟧
    BATH    (  002     , 16      , yes )        /* Rinsing in hot water out*/
      ⟦ LEVEL_SENSOR STEAM_VALVE ⟧
    BATH    (  008-01 , 17       , yes )                       /* Corroding */
      ⟦ RECTIFIER("TBPI-800/12")
      LEVEL_SENSOR
      STEAM_VALVE
      RESERVOIR ⟧
    BATH    (  008     , 18      , yes )                       /* Corroding */
      ⟦ RECTIFIER("TBPI-1600/12")
      LEVEL_SENSOR
      STEAM_VALVE
      RESERVOIR ⟧
    ⟧
```

4.1.3. *The Transformation Process*

During the transformation process (see Fig. 6), a SyntheCAD document is transformed into a set of nodes with attributes (a logical tree). The obtained logical tree is used further as input to the formatting process.

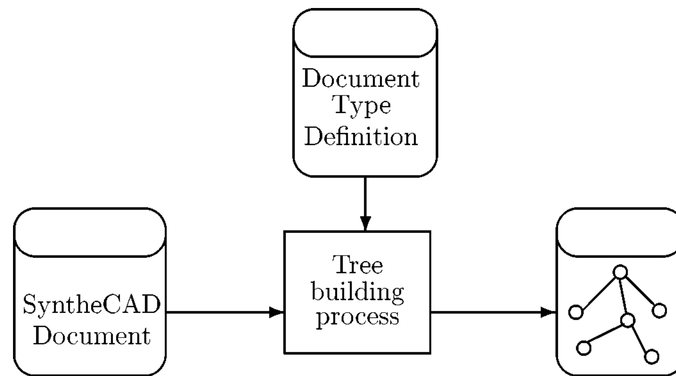

Fig. 6. The conceptual model of the transformation process.

All operations performed in the transformation process are independent of the formatting process. During the transformation process, syntax analysis of a SyntheCAD document is performed. When a node type and types of the attributes accord with the DTD, following operations are performed:

- creating nodes and attributes, according to user specification;
- creating the attributes, which are not mentioned in the user specification, but are specified in arrangement template.
- creating given relations between the nodes.

4.2. *The Style Language*

The style language specifies the formatting process (see Fig. 7). The formatting process applies presentation styles to source document content. The style language defines the visual appearance of a formatted document in terms of formatting characteristics attached to an intermediate tree.

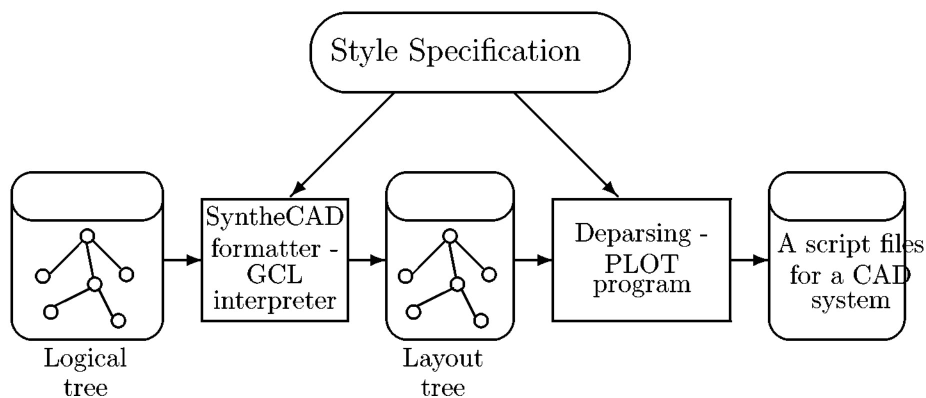

Fig. 7. The conceptual model of formatting.

The formatting process uses a style-specification, which may include construction rules, page-model definitions and other application-defined declarations and definitions.

4.2.1. *The Formatting Process*

The formatting process consists of the following subprocesses:

- application rules to the objects in the logical tree;
- definition of the page geometry;
- the layout of the content based on the rules specified by the semantics of object class and the values of the characteristics associated with these objects. Each object (an instance of an object class) is formatted in order to produce a layout and is positioned by a parent in the object tree.

The conceptual model consists of the formatting actions:

1. enrich the logical tree with semantic information;
2. apply construction rules to the component nodes;
3. calculate the layout structure;
4. make semantic analysis of layout structure, i.e., provide pagenumbering, make piping, cabling, etc.;
5. prepare output code for a user-defined CAD system.

4.2.2. *The GCL Language*

High level declarative language GCL (Graphical Circuit Language) was developed for specifying and creating layouts. According to the principles of silicon compilation, the language serves for knowledge representation and 2D graphics. It has a syntax and 2D graphics features of ICL, i.e., a silicon compilation language (Ayres, 1983). A GCL function is associated with each component type.

One way to create the layout is to specify it in a language which admits convenient specification of geometric items and their combinations. All suitable languages for layout specification have in common notations for specifying two-dimensional points. (Ayres, 1983) provide an introduction to the silicon compilation and the art of automatic chip generation.

The following types come built in: INT, REAL, BOOL, TEXT, POINT. The types are created and examined by obvious in other language' operations. For a POINT type there are also the following less obvious operations:

POINT	`\MAX`	POINT	→	POINT	[e.g., (3#4) `\MAX` (6#2) is 6#4]
POINT	`\MIN`	POINT	→	POINT	[e.g., (3#4) `\MAX` (6#2) is 3#2]
POINT	=	POINT	→	BOOL	(the points are equal)
POINT	<>	POINT	→	BOOL	(the points are not equal)
POINT	<	POINT	→	BOOL	(the first point lies to the left and below the second point)
POINT	>	POINT	→	BOOL	(the first point lies to the right and above the second point)

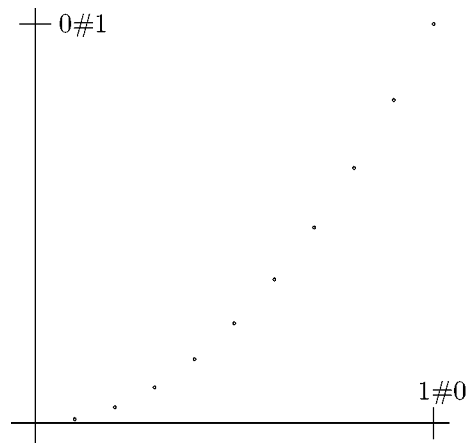

Fig. 8. Points in a parabola.

Example. An expresion `A#A*A FOR A FROM 0 TO 1 BY 0.1` denotes 11 points: `0#0, 0.1#0.01, 0.2#0.04, 0.3#0.09, ...,1#1`, which happen to trace out the piece of parabola (Fig. 8).

Declaring New Types

To declare new types we use the "TYPE" statement:

TYPE name-of-new-datatype = a-type-expression

For data type COLOR the type declaration is

TYPE COLOR = TEXT;

then we can create the variables:

VAR RED: COLOR ;

RED := "RED" ;

It is possible to construct new data types of all sorts. A polygon type is represented as an sequence of points:

TYPE POLYGON = { POINT };

Declaring, Creating, and Examining Lists

{ point ; point ; ... ; point } → POLYGON

{ layout ; layout ; ... ; layout } → LAYOUTS

Example:

{ 0#0 ; 3#5 ; 7#2 ; 4#1 } → POLYGON

We can also append and concatenate lists:

element `<$` list → list (i.e., left append)
string `$>` list → list (i.e., right append)
string `$$` list → list (i.e., concatenation)
REVERSE (list) → list (i.e., reverses the order of a list)

The list examines quantifier (linguistic name for loop operator):
FOR variable `$E` list

We can also produce the list by writing
{ COLLECT element Quantifier }
For example,
{ COLLECT X#X*2 FOR X FROM 1 TO 5; }
(point) (quantifier)
is equivalent to { 1#2; 2#4; 3#6; 4#8; 5#10 }.

Records

We declare record data type by
[selector : type selector : type ... selector : type]
A new type BOX is declared by
TYPE BOX = [LOW: POINT HIGH: POINT];
For BOX we can define the function `\TO`:
DEFINE TO(A, B: POINT) = BOX : [LOW: A HIGH: B]
ENDDEFN
Thus, 0#0 `\TO` 1#1 also means a box with a lower-left corner 0#0 and upper-right corner 1#1. So, we have two operations:

- one creates a box as a single polygon:
{ $POINT_1; POINT_2; POINT_3; POINT_4$ } `\PAINTED` COLOR
- another creates a single box:
$POINT_1$ `\TO` $POINT_2$ `\PAINTED` COLOR

Variants

This type constructor gathers together unrelated types to form a new type which admits a well-defined uncertainty in representation. Now we can define the type LAYOUT by writing:

```
TYPE LAYOUT = EITHER
                     BOX      = BOX
                     POLYGON  = POLYGON
                     UNION    = LAYOUTS
                     COLOR    = [PAINT: LAYOUT WITH: COLOR]
                     MOVE     = [DISPLACE: LAYOUT BY: POINT]
                  ENDOR ;
```

With a variant obiect are inserapable the CASE statements:

CASE variant object OF
$STATE_1$: analyze (certain object)
$STATE_2$: analyze (certain object)
ENDOR;

About Quantifiers
A "quantifier" is a loop generator. Three quantifiers are introduced in GCL: FOR, WHILE, REPEAT. The following quantifier operators are introduced to form a new quanitifier:

INIT – an action occurs before the quanifier starts up,

FIRST_ DO – the action is performed only upon the first iteration,

OTHER_ DO – the action is performed on all nonfirst iteractions,

EACH_ DO – we can append the quantifier that is to occur upon each iteration. The OTHER_ DO complements FIRST_ DO, and together, these two clauses form the equivalent of an EACH_ DO.

WITH – we can filter out those iterations for which the boolean expression yields FALSE,

&& – a new quantifier is formed from two given quantifiers by repectively lock-stepping two. The action is performed, while the boolean expressions of both quantifiers produce TRUE,

!! – a new quantifier is formed from two given quantifiers by repectively nesting two (Cartesian product),

THERE_ IS – for expressing the conditions like this: there is an X in A, and Y in B such that X \EQ Y. We write this condition in GCL with
THERE_IS X \EQ Y FOR X $E A; !! FOR Y $E B;
or with
FOR X $E A; !! FOR Y $E B; THERE_IS X \EQ Y

Standart Unit Specification
Each component is defined by a function:

```
DEFINE  name ( arguments ) = type:
        BEGIN
                VAR variables_if_needed
                DO      actions_if_needed
                GIVE    value_of_the_type_described_above
        END
ENDDEFN
```

A layout of a unit we define with the function notation DEFINE ... ENDDEFN. When the output type is LAYOUT, we become the layout of the unit.

Complex Layouts
Since a layout is a picture, we can conceive that the layout is made up of a large set of individual polygons. Each polygone represents not just the outline of the polygon, but the entire area enclosed by the polygon. Each polygon area has a colour, too. But designer rarely thinks in terms of pure polygons. He/She thinks in terms of functional units made up of small sets of polygons.

A practical layout consists of more than a single polygon or box. There are two equivalent notations for specifying the superposition of existing layouts:
1 $LAYOUT_1$; \UNION $LAYOUT_k$
2 { $LAYOUT_1$; $LAYOUT_2$; ...; $LAYOUT_k$ }

4.3. *Plotting a Layout*

A plotting process interfaces between a layout and specific plotter. One of user requiments is that plotting process must be easy to modify. Now, we can describe the plot program :

```
DEFINE PLOT ( L:LAYOUT   DISP:POINT   COLOR:COLOR);
   BEGIN   VAR   L1   =   LAYOUT;
      CASE   L   OF
         POINT:          plot the point displaced by DISP, with COLOR
         BOX:            plot the box displaced by DISP, with COLOR
         CIRCLE:         plot the cicle displaced by DISP, with COLOR
         TEXT:           plot the text displaced by DISP, with COLOR
         POLYGON:        plot the polygon displaced by DISP, with COLOR
         UNION:          FOR L1 $E L; DO
                                             PLOT (L1, DISP, COLOR)
                                          END;
         MOVE:           PLOT ( L.DISPLACE, DISP+L.BY, COLOR );
         COLOR:          PLOT ( L.PAINT, DISP, L.WITH );
      ENDCASE
   END
ENDDEFN
```

This recursive program is easy to understand. For a complex layout, the function recursively calls itself until a graphical primitive (point, line, circle and so on) is obtained. In order to change a CAD tool, the user has to modify a PLOT program.

4.4. *Data Flow in SyntheCAD*

In organisational context, we have chosen a completely automatic configuration. It makes the configuration capable to people without the required background knowledge and reduces the workload of experienced users.

In Fig. 9 the data flow in SyntheCAD is shown. The logical structure (arrangement) of an electroplating line is given by the user and serves as an input. In an arrangement the key components of a particular electroplating line are given.

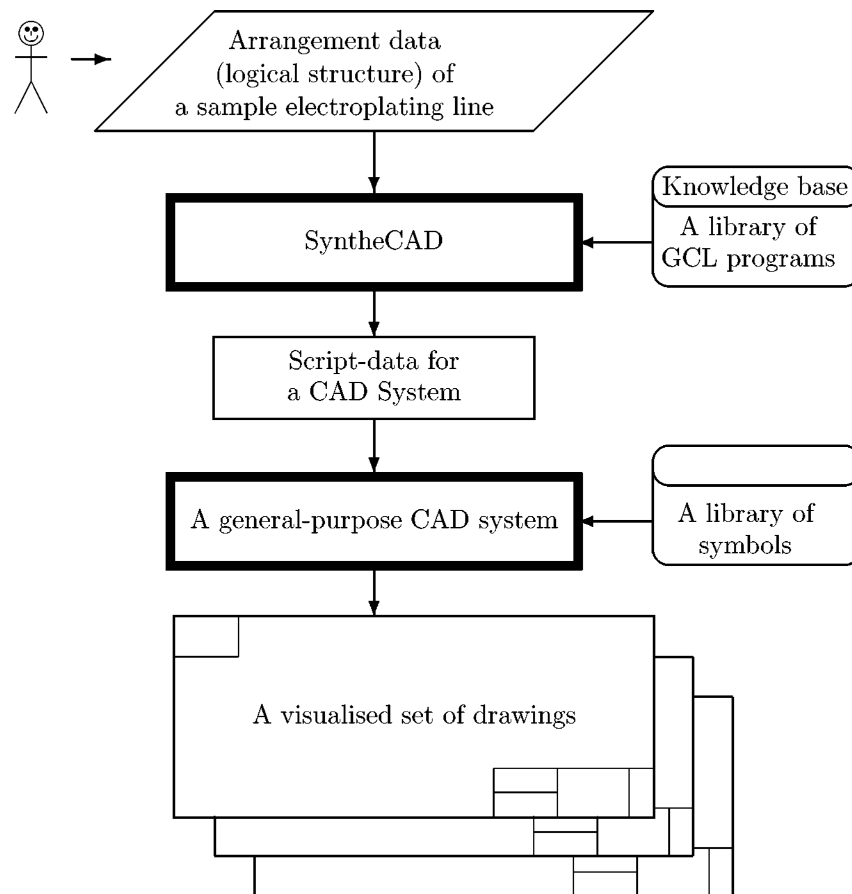

Fig. 9. Data flow: from a customer's order to drawings.

The script files of technical drawings for a general-purpose CAD system serve as output. A configurator's knowledge base and a library of symbols is prepared before configuring.

4.5. *Implementation of Component Oriented Method*

We have mentioned three basic stages, which are performed in component oriented approach (Section 3.2).

The logical structure (arrangement) defines what components are created. It is the first stage in the component oriented configuration process. Component types are chosen by the user.

The second and third stages of the configuration are performed during the interpretation of the logical tree. In this step the following sub-steps are performed: forming component's layout, connecting the components, cabling, dividing the drawings into pages, etc.

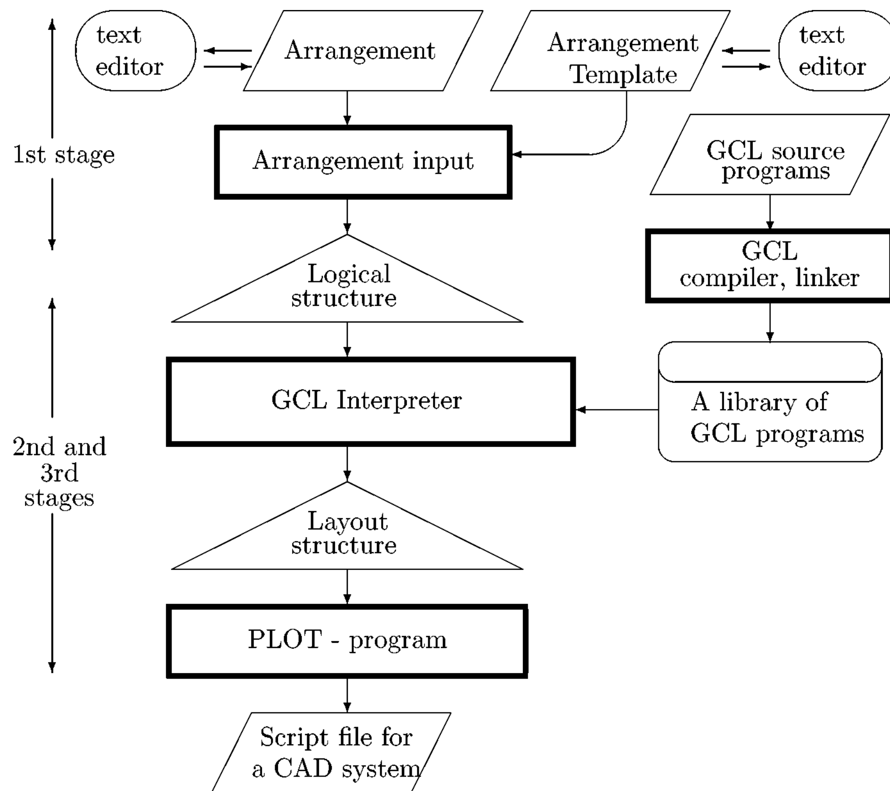

Fig. 10. Three configuration stages in SyntheCAD.

After the interpretation, the layout structure is derived from the logical structure. Actions are specified in GCL programs.

The architecture of SyntheCAD consists of arrangement input, GCL interpreter, GCL compiler, linker, and a PLOT program (Fig. 10). During the arrangement input components are created. A logical parser-tree of an electroplating line results. The interpreter traverses the tree and adds new variables (i.e., components and attributes) to the configuration, enriches the logical tree with semantic information, calculates the layout structure, produces semantic analysis (i.e., divides into pages, connects components, makes cabling, etc.). The main goal in the interpretation step is to yield the layout structure of the drawings.

5. Conclusions

The aim of this paper is to present a method of automatic graphical document preparation as it is implemented in the SyntheCAD system developed at Vilnius University. In our domain the logical structure of drawings constitutes the essence of the domain model.

The method proves, that logical structure of graphical documents can be formalized and used further in the configuration.

The document preparation system SyntheCAD is designed for specifications which are applicable to 2D graphical documents, that have an internal aggregation structure. SyntheCAD uses essentially the logical structure of documents to be synthesized. In our case technical documentation is produced.

References

Ayres, R.F. (1983). *Silicon Compilation and the Art of Automatic Microchip Design*. Prentice-Hall, Englewood Clifs, New Jersey.

Brown, David C. (1997). *S*ome thoughts on configuration processes. http://www.cs.wpi.edu/ dcb/Config/configuration.html

Čyras V., V. Povilaitis, D. Sidarkevičiūtė, K. Moroz (Lapin), M. Žilinskas, R. Baracevičius (1993). A Model-driven configuration: from procedural to object-based decomposition. In M.R. Beheshti and K. Zreik (Ed.), *Advanced Technologies*, Elsevier Science Publishers B.V.

Furuta R., J. Scotfield, A. Shaw (1992). Document formatting systems: survey, concepts and issues. In J. Nievergelt, G. Coray, J.D. Nicoud, A.C. Shaw (Eds.) *Document Preparation Systems*, North-Holland Publishing Company.

Eickel, J. (1990). Logical and layout structures of documents. *Computer Physics Communication*, **61**,201–208.

Gajski D.D., D.E. Thomas (1988). Introduction to Silicon compilation. In *Silicon Compilation*, Addison-Wesley, pp. 1–48.

ISO-Standart 10179 (1996). *Information Technology – Processing languages – Document Style Semantics and Specification Language (DSSSL).*

ISO-Standart 8879 (1986). *Information Processing – Text and Office Systems – Standart Generalised Markup Language (SGML)*.

Maler, E., J.E. Andaloussi (1996). *Developing SGML DTDs: From Text to Model to Markup*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458.

Mittal, S., F. Frayman (1989). Towards a generic model of configuration tasks. In *Proc. of the IJCAI'89*, pp. 1395–1401.

Stumptner, M. (1997). An overview of knowledge-based configuration. *AI Communications*, **10**, 111–125.

Wilhelm, R., R. Heckman (1996). *Grundlagen der Dokumentenverarbeitung*. Bonn, Addison-Wesley-Longman.

K. Lapin has received her Diploma in Computer Science from Vilnius University in 1988. Since then she is a researcher at the Faculty of Mathematics of Vilnius University. Her research interests include document processing, document structure representation, knowledge-based configuration.

Struktūrinių grafinių dokumentų automatinis konfigūravimas galvaninių linijų srityje

Kristina LAPIN

Pristatomas techninių brėžinių automatinis konfigūravimas galvaninių linijų dalykinėje srityje. Aprašoma, kaip sukurtoje sistemoje SyntheCAD integruojami du metodai: dokumentų ruošimo srityje pasiūlytas *apibendrintas kodavimas* bei konfigūravimo uždavinių klasėje naudojamas *komponentinis metodas*.