THE INCOME APPROACH FOR CONCEPTUAL MODELLING AND PROTOTYPING OF INFORMATION SYSTEMS

G. Lausen *)
T. Németh **)
A. Oberweis *)
F. Schönthaler **)
W. Stucky **)

*) Universität Mannheim
Fakultät für Mathematik und Informatik
D-6800 Mannheim
West-Germany

**) Universität Karlsruhe (TH)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
D-7500 Karlsruhe
West-Germany

ABSTRACT

This paper surveys the main features of INCOME, which is an approach for conceptual modelling of information systems. INCOME supports the specification and prototyping of all static and dynamic system aspects which are regarded to be relevant for the design. Petri nets with different interpretations are used as uniform specification framework for object structures and system behaviour.

KEYWORDS

Conceptual Modelling, Prototyping, Predicate/Transition Net, Semantic Hierarchy Object Model.

I INTRODUCTION

Conceptual modelling is the activity of formally specifying all relevant aspects of an application system in such a way that implementation aspects are not yet regarded. In a *conceptual schema* both static and dynamic aspects should be considered (cf. [ISO82]). The conceptual modelling step usually is placed between the requirements analysis step and the system design step.

INCOME (Interactive Net-based COnceptual Modelling Environment) is a computer-supported approach for conceptual modelling. Its characteristic feature is the uniform framework based on Net Theory (cf. [BRR87]) to capture all relevant static and dynamic aspects of the future application system. Moreover, to allow a validation of the information content and the application procedures as early as possible INCOME provides a prototyping facility that can be used at any stage of the design process.

Input for the conceptual modelling with INCOME is a *functional requirements specification* given as a hierarchy of object flow diagrams similar to SADT [Ros77] or ISAC [Lun82]. Additionally a glossary is needed which is an informal textual description of functions and object flows of the future system. From the functional requirements specification the *conceptual object structure schema* is derived. We use a semantic hierarchy object model similar to SHM+ [BrR84] and THM [Sch84]. The object flow diagrams of the hierarchy are interpreted as Petri Nets. If necessary, these *local behaviour nets* are modified so that the formal transition rule holds. This formal transition rule makes the modelling of system dynamics possible. The local behaviour nets are combined to the *global behaviour net* which expresses system dynamics on the user level. The *transaction schema* contains partial nets of the global behaviour net together with arc inscriptions to describe what object types are involved in a given transaction and how they are used.

In [OSV82, OST83, BrR84, KiM84, AnL85, SoK86, StH86, HNS87] similar comprehensive methodologies for conceptual modelling are described where static aspects of application systems as well as dynamic aspects are considered. However, it is not clear how the formal system specification can be derived in these approaches from the usually informal requirements description given by the system enduser. Based on these approaches, communication between endusers and designers would be rather troublesome because different description formalisms and graphical representation methods are used for the static and dynamic partial schema.

In INCOME, on the other hand, *Petri Nets* are used for the uniform specification of both static object structures and dynamic system behaviour. INCOME provides concepts for a stepwise derivation of formal specifications from informal descriptions. Using concepts of refinement for nets allows the designer to consider static and dynamic system aspects at more or less detailed levels of abstraction. Automatic *analysis tools* allow the checking of a proposed conceptual schema for syntactic errors whereas the *prototyping tool* helps to detect design errors like incompleteness, contradiction, ambiguity, and to improve the design in cooperation with the enduser.

The basic concepts of INCOME have been first proposed in [Lau86, Lau88]. The prototyping tool has been described in [SOL87, Sch89]. Other special aspects of conceptual modelling have been considered elsewhere: The representation of temporal aspects in [ObL88], and the specification of integrity constraints in [Obe88]. Earlier overviews of the INCOME approach can be found in [OSL86, NSS88]. In this paper the INCOME approach is examined in an extensive case study on an inventory control and purchasing system. However, due to space limitations we must refer to [LNO88] for most parts of the case study.

This paper is organized as follows: The functional requirements specification with object flow diagrams is described in Section II. The procedure of object structure modelling is outlined in Section III. Section IV briefly explains how system behaviour is specified in terms of PROLOG-inscribed

Predicate/Transition Nets. The INCOME prototyping approach is described in Section V. The prototype implementation of a software development environment based on the INCOME approach is outlined in Section VI. Section VII contains a short summary of the paper and an outlook on future work.

II Functional Requirements Specification

The starting point for conceptual modelling with INCOME is a functional requirements specification given as a hierarchy of object flow diagrams. *Object flow diagrams* are an easily understandable means for the semi-formal description of functions that an application system must fulfil. Only two graphical symbols are needed: rectangles for the representation of functions and arrows between rectangles to represent object flows between the respective functions. Hierarchies of object flow diagrams are derived by successive refinement of object flow diagrams where refining an object flow diagram means replacing each single function together with its input and output object flows by a complete object flow diagram. The top level function of a hierarchy represents the whole system activity whereas the bottom level functions, called final functions, represent atomic user operations. Figure 1 shows an example of a refining object flow diagram.

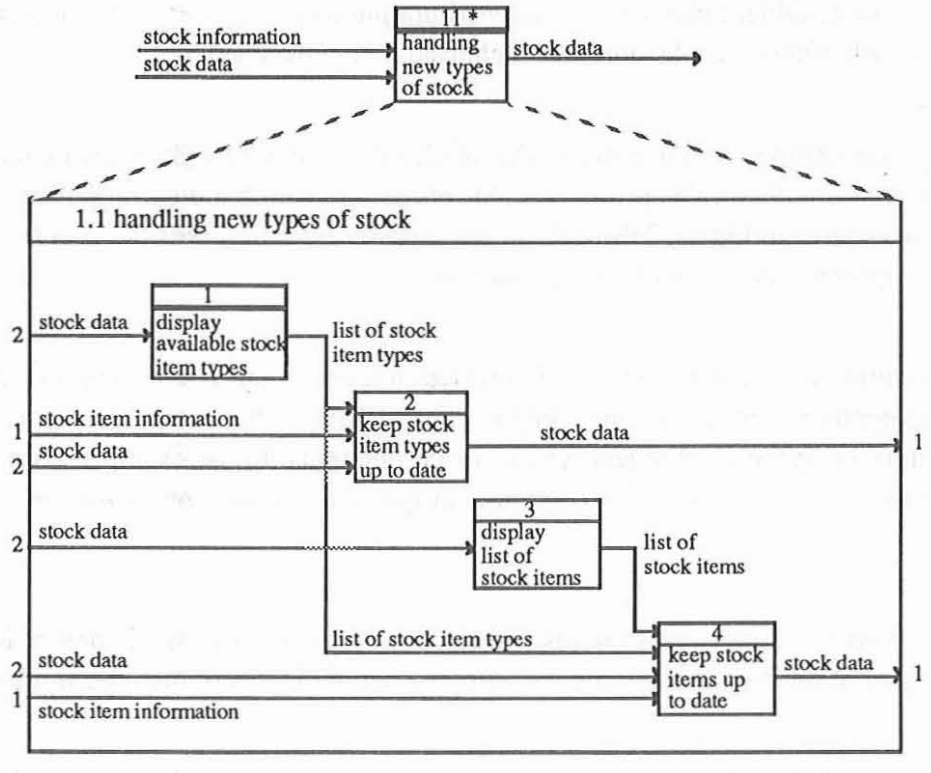


Figure 1: Refining Object Flow Diagram

The top-down approach supported by INCOME enables a trained application system enduser to specify his requirements himself. However such a semi-formal system description is not an appropriate basis for a later system implementation because contradictions, inconsistencies, incompleteness, lack of exactness, and ambiguity cannot be excluded by automatic analysis. We reject to introduce further semantics in object flow diagrams by using additional graphical symbols as done for example in [War86] because for

practical applications such extended flow diagrams become too complicated and are therefore hardly understandable for system endusers. Instead we prefer an approach where we proceed in a stepwise manner from semi-formal object flow diagrams and informal textual descriptions within the glossary to the formal specification means of Petri Nets. Petri Nets and the underlying Net Theory [BRR87] are based on mathematical formalisms and allow automatic analyses as well as prototyping-based validation of the system specification.

III Modelling of Object Structures

III.1 Semantic Hierarchy Object Model

The static part of the conceptual schema - the object structure schema - contains descriptions of objects and relationships of a real or hypothetical world. For the modelling of object structures we use nets with a special interpretation, so-called *object structure nets*. The underlying concepts of classification, aggregation, generalization, and grouping are well known to be fundamental for most semantic data models (e.g. [SmS77, BrR84]).

Classification

Classification is the essential concept for object structure modelling. It is used to describe objects in terms of object classes. Object classes are assigned unique names (types). The objects of a class are called *instances* - each of them can be uniquely identified within the class.

Generalization

The concept of generalization is comparable to that of classification where generalization is used to collect object types (*subtypes*) with similar properties. All properties which are common for the subtypes are assigned to the *supertype* . In Figure 2 the object type *supplier information* is the generic supertype of the subtypes *address information* and *delivery information.*

Aggregation

Whereas generalization allows the insertion of abstraction levels, aggregation supports the structuring of object types. Aggregation is used to define component relationships between object types. A set of object types is assigned to the aggregated object type as its components. In the example given in Figure 2 the aggregated type *stock item* consists of the component types *item code*, *item name*, *maximum level* and *re-order level.*

Grouping

Grouping is used to define object types whose instances are sets of objects of another lower level type. In Figure 2 an object of the type *delivery information* is a set of objects of the type *stock item.*

Inheritance

An essential feature of the object model is the concept of property inheritance (cf. [BrR84]). The inheritance direction is given by the direction of the arcs in the graphical representation. In this paper only domain inheritance is considered. The *domain* of an object type may be either elementary (CARDINAL, INTEGER, REAL, STRING, ...) or composed of inherited domains.

Connect Operator

Modelling of object structures is based on a connect operator which is used for the formal connection of two object structure nets. The resulting net is computed as the union of the underlying sets of object type relationships.

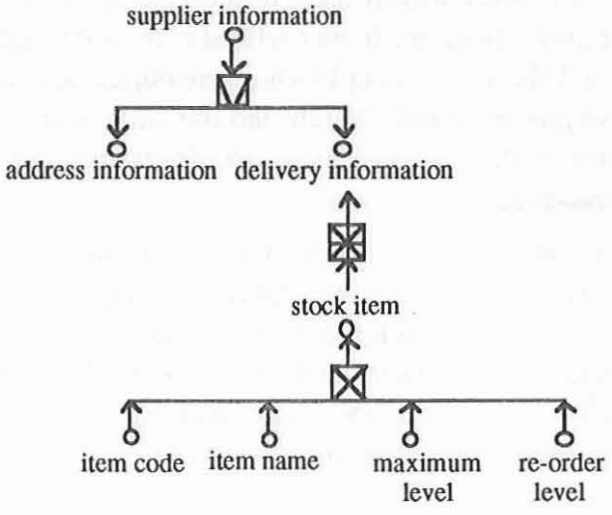


Figure 2: Example of an Object Structure Net

III.2 LOCAL STRUCTURES

Object structure modelling starts with the interpretation of object flows of the object flow diagram hierarchy as object types. This interpretation cannot be fully automatized because the naming of object flows in the functional requirements specification depends on the context. Renaming and insertion of object types have to be done interactively. The resulting set of object types is a framework for further object structure modelling.

In a second step local object flow relationships such as predecessor-/successor- and refinement-relationships are mapped into the object model. The formal mappings reflect the results of several case studies carried out in different application areas (cf. [NSS88]).

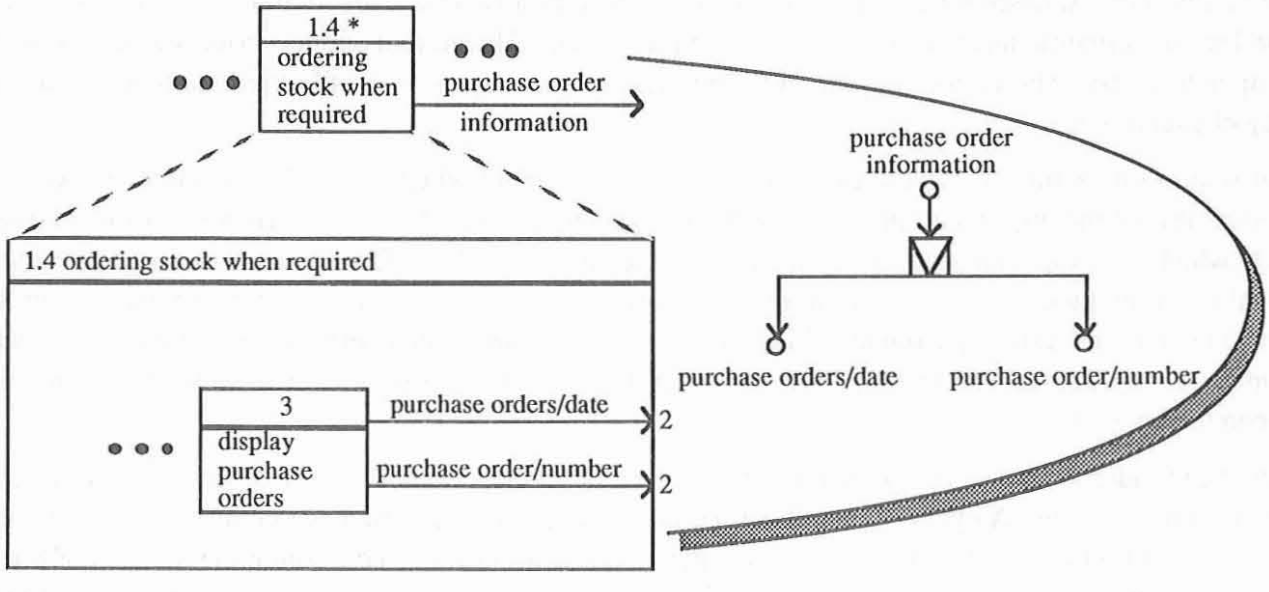


Figure 3: Local Structure for a Further Refined Object Flow

Modelling of local structures is done with respect to the type of the underlying object flow. Local structures relate those object flows which are further refined in the object flow diagram hierarchy to their detailing object flows. Figure 3 shows an example where the object flow *purchase order information* is further detailed by the flows *purchase orders/date* and *purchase order/number*. The relationship is mapped into a generalization with supertype *purchase order information* and subtypes *purchase orders/date, purchase order/number.*

Local structures referring to output object flows of functions which are not further refined (final functions) are derived from predecessor- and successor-relationships. Output object flows are related to input flows and other output flows of the corresponding function. Figure 4 shows an example where the output flow *stock data* of function *keep stock items up to date* is related to the input flows *list of stock items, list of stock item types, stock data,* and *stock item information.* Note that in this example additional information was inserted into the final local structure.

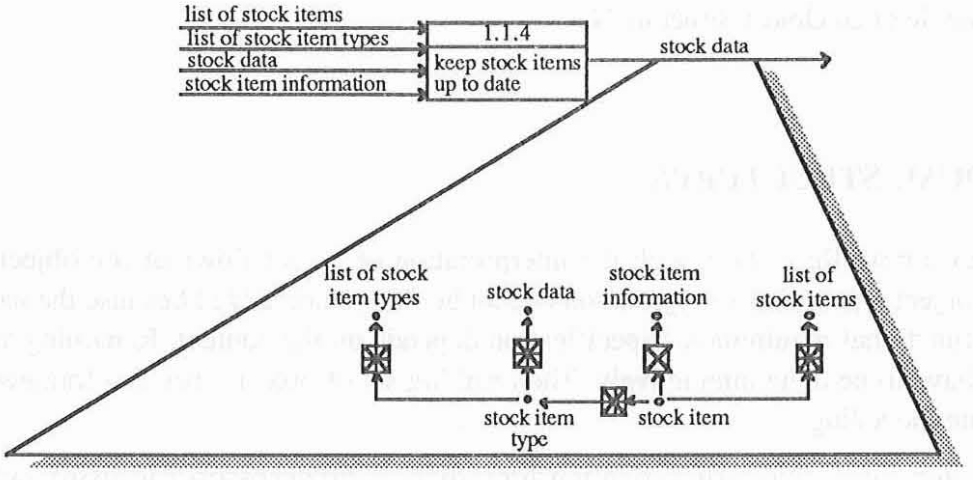


Figure 4: Local Structure for an Output Flow of a Final Function

Most part of the modelling process is interactive depending on the depth of refinement realized in the hierarchy. Formal mapping of object flow relationships usually only leads to draft proposals because of the lack of semantic information in the diagram hierarchy. Hence tool support covers also a powerful graphical editor. The editor supports different display techniques to enable the handling of complex object structure nets.

An important feature of the editor is the analysis component which is used to support the semantic correctness of the object structure nets. A specific situation to avoid is the occurrence of isolated partial nets which may especially occur when editing large structures. Therefore object structure nets must be weakly connected. Isolation often results from removing a supertype and the corresponding generalization relationship from the object structure net. To ensure that an object structure net is weakly connected, appropriate algorithms have to be applied which are well known from the relevant graph theoretic literature.

Another fundamental presumption for the existence of a correct object structure net is the absence of cycles within the net. A cycle is given by a sequence of object types which are connected by means of a directed path with respect to the arcs of the object type relationships. This condition is essential for the semantic correctness because the domains of object types within a cycle cannot be determined. Cycles within structures are detected by automatic analyses.

It is a general principle of the INCOME tool support that proving the correctness of specifications is not rigorously enforced by a certain design procedure. Therefore temporary incorrectness is permitted. The designer explicitly defines the moment when correctness of the object structure net is demanded. This decision results in an activation of the respective analyses.

III.3 SCHEMA PARTS

Predecessor- and successor-relationships on a higher hierarchy level are the subject of the third step of object structure modelling. Further refined output object flows are related to input flows of the corresponding functions. Object structure nets resulting from this step are called *schema parts*.

An assumption of schema part modelling for a certain object flow O is that all schema parts and local structures required for the object flows of the lower level diagram are present. Depending on the strategy mentioned below, these local structures and schema parts are successively connected with the local structure of object flow O. The connection is done by application of the operator described in Section III.1. Because in general most of the local semantics necessary for the definition of object types is introduced in the local structuring step, most steps of schema part modelling can be automatized.

The strategy for schema part modelling of object flow O is as follows:

(1) The set of object flows detailing the higher level object flow O is determined.
(2) For each of the object flows of step (1) paths are computed backwards through the diagram with respect to predecessor- and successor-relationships.
(3) All schema parts and local structures related to object flows contained in any of the paths computed in step (2) are connected successively to the local structure of the object flow O.

Schema parts constructed in this way may now be used for schema part modelling of object flows on a higher hierarchy level.

The procedure described in the previous sections results in one schema part for each output flow of the top level function. In order to achieve one global object structure schema, the next step is to integrate these schema parts in one common schema. This schema has to be augmented by those local structures and schema parts which have not yet been considered in the schema.

Schema integration is done with respect to the formal connect operator. Moreover, the design is supported by additional facilities already described in [NaG82].

IV MODELLING OF SYSTEM BEHAVIOUR

The behaviour schema formally describes system behaviour. System behaviour is considered on two different levels: on the user and the database level. Usually the system enduser does not directly apply atomic database operations (like *delete*, *update*, *insert*), but instead applies user operations, so-called *transactions* which are composed of atomic database operations.

In the first step of behaviour modelling, the object flow diagrams of the functional requirements specification are interpreted as local behaviour nets. The resulting nets are modified and sometimes further refined to achieve the validity of the formal transition rule for the bottom level nets. In a next step the *local behaviour nets* are connected to a *global behaviour net* which represents all relevant user operations and the object flows between them.

The elementary user operations are further refined by so-called *transaction nets* which are given as PROLOG-inscribed Predicate/Transition Nets [NiV86].

IV.1 LOCAL BEHAVIOUR NETS

The object flow diagrams of the functional requirements specification serve as a first overview of the functions of the future application system. They are not a suitable means for a complete and formal specification of system dynamics.

Petri Nets, on the other hand, are a widely accepted and well suitable means for a formal description of such aspects. They allow the representation of concurrent, alternative, and sequential processing of transactions. Marking of places with tokens and the definition of the formal transition rule make the description of system dynamics possible: Objects that move through a system from activity to activity are represented in Petri Nets as tokens that move between places.

In the first step of behaviour modelling, the object flow diagrams are interpreted as Petri Nets: the functions are interpreted as transitions, the object flows are interpreted as places together with outgoing or ingoing arcs of the corresponding transitions. Figure 5 shows the Petri Net which is derived from the object flow diagram *1.1 / Handling New Types of Stock* in Figure 1.

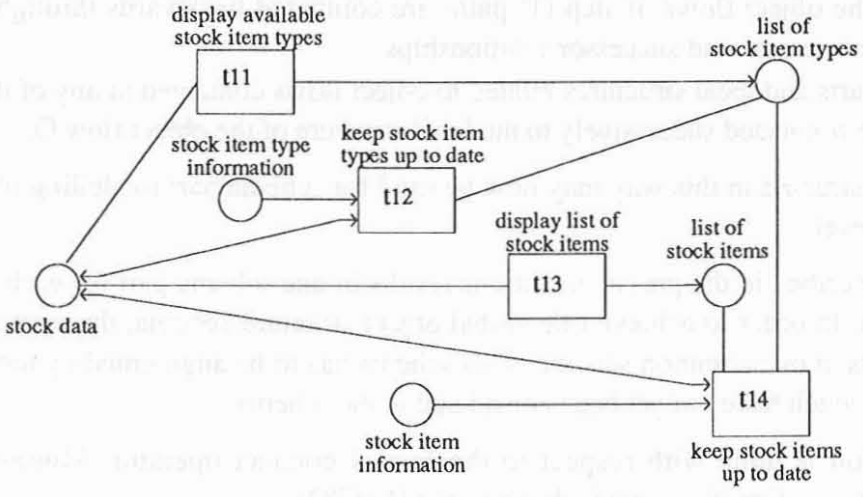


Figure 5: Interpretation of Object Flow Diagram *1.1* as a Petri Net

The result is a hierarchy of nets where the formal transition rule usually does not yet hold. The INCOME method requires a net hierarchy where the bottom level nets represent system behaviour with respect to the transition rule. For this, first the bottom level nets are modified and if needed further refined. In a bottom-up procedure the higher level nets must be adapted to the modified lower level nets. Usually this adaptation results in a modified object flow diagram hierarchy.

The question whether a behaviour net represents correct system behaviour and which modifications are necessary can only be decided interactively. This process is supported by information about the object structures corresponding to the places.

INCOME supports reachability analysis of local behaviour nets, for further information see [OSS87]. Furthermore, generation of a reachability tree makes detection of deadlocks possible.

IV.2 GLOBAL BEHAVIOUR NET

Most part of the construction of a global behaviour net can be automatized. The construction is done by connecting the local behaviour nets in a top-down process.

The algorithm works as follows (a more detailed description is given in [Lau86]): Surroundings in a net (transitions together with the corresponding input and output places and arcs) are replaced by their refinement until all nets of the hierarchy are considered. The connection algorithm preserves the behaviour of the global net with respect to the intended behaviour of the local nets. Replacing requires to consider places which share the surroundings of the refined transitions. If the refinements of those places in different nets are equal, the connection is trivial.

The other cases are non-trivial. It must be distinguished between two cases:

Case 1: The object type corresponding to the higher level place and the object type of the lower level places are interrelated by a component relationship. This corresponds to an aggregation structure.

Case 2: The object type of the higher level place and the object types of the lower level places are interrelated by a subset relationship. This corresponds to a generalization structure.

Integration of these object modelling concepts with Petri Nets leads to the following net interpretations:

Aggregation: One token of the higher level place corresponds to an assignment of one token to each lower level place.

Generalization: One token of the higher level place corresponds to an assignment of one token to exactly one lower level place.

This object modelling oriented view on pre-images of places now can be used for the derivation of the global behaviour net. First a local behaviour net is transformed to an augmented local behaviour net as follows:

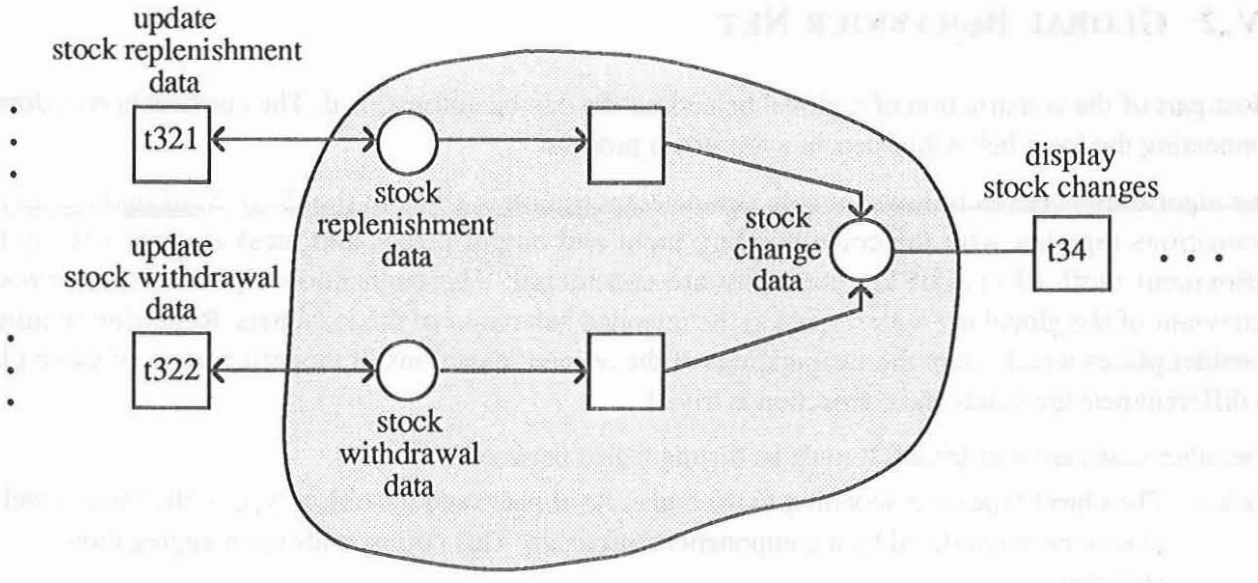
(1) Each in-set of a place, which contains more than one place, is replaced by either a decomposition or a specialization net.

(2) Each out-set of a place, which contains more than one place, is replaced by either a composition or a generalization net.

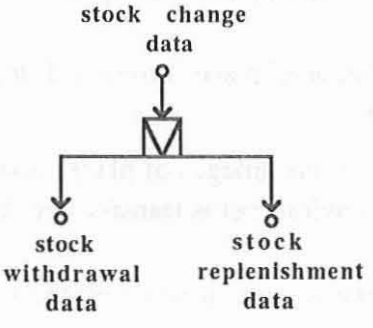
The connection is then done by replacing the surroundings by the corresponding augmented local behaviour nets. Figure 6 illustrates an example of the case study where the connection is done by means of a generalization net.

IV.3 TRANSACTION NETS

In the global behaviour net system behaviour is only considered on the user level, i.e. on transaction level. The formal Petri Net transition rule allows the representation of pre- and post-conditions of transactions in a way such that each input place must be marked with at least one token and the capacity of the output places must not be exceeded. However, the tokens flowing through the behaviour net are anonymous objects which are not distinguishable from each other if they are in the same place. Hence it is not possible to specify pre-conditions with respect to concrete instances of objects. Moreover, there is no possibility to specify how the output tokens are derived from the input tokens.



(a) Connection of Local Behaviour Nets



(b) Part of the Object Structure Schema

Figure 6: Connection of Local Behaviour Nets using a Generalization Net

The specification of this information is done in the transaction modelling step. The surrounding of each final transition is further refined by representing it in terms of a PROLOG-inscribed Predicate/Transition Net. To the arcs formal sums of variables are assigned where each variable is associated with an object type of the object structure schema. To the places object types are assigned where the object types of the corresponding arcs are equal to that or are connected with it as the subtype of a respective generalization. The transitions are inscribed with PROLOG clauses.

Figure 7 shows an example of a transaction net derived from the surrounding of transition *t12 / Keep Stock Item Types Up to Date*. The transition inscription specifies how, depending on certain rules, information about an object type is updated or inserted in the stock data.

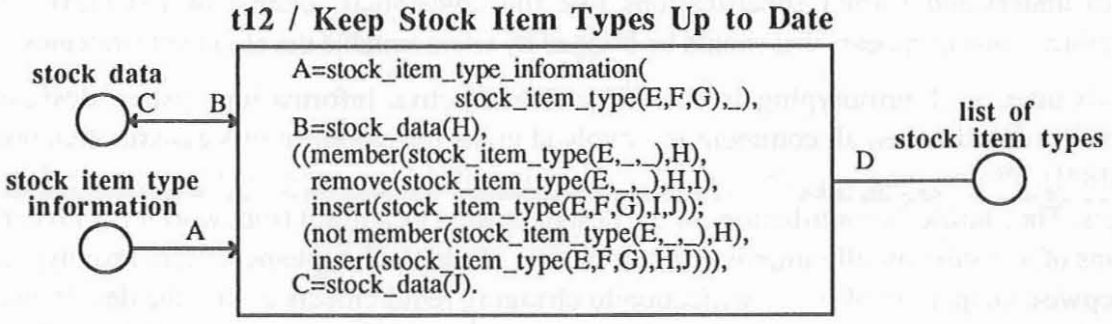


Figure 7: Transaction Net *Keep Stock Item Types Up to Date*[1].

IV.4 Specification of Integrity Constraints

The specification of integrity constraints is an important step during conceptual modelling which concerns both static and dynamic aspects. Static integrity constraints restrict the set of system states whereas dynamic integrity constraints restrict the set of state transitions.

Static integrity constraints must be modeled for each transaction net because transactions change system states and possibly violate integrity. Following the proposals of [HeR86, Vos87] we model static integrity constraints by facts which are transitions that are postulated to be never enabled. Facts represent (negative) assertions about admissible system states because they restrict the set of possible states to such states where no fact is enabled. The facts are inscribed with PROLOG clauses that represent violations of the integrity constraints. More information about this concept can be found in [Obe88]).

Dynamic integrity constraints concerning the order in which objects are created, deleted, or manipulated must be guaranteed by the structure of the behaviour net. If e.g. an object a which is created by a transition T_a must exist before another object b can be created by a transition T_b, then there must exist an object flow from transition T_a to T_b in the behaviour net. Other dynamic integrity constraints concern absolute clock times or calendar dates.

In Petri Nets those temporal aspects are usually not considered, i.e. transition occurrences have no duration and the tokens' temporal availability is not restricted. Especially in office environments temporal aspects like durations of activities, starting times of activities, time limits and availability times play an important role. We use a clock based method first introduced in [Ric85] to model temporal restrictions in Predicate/Transition Nets without leaving the framework of Net Theory. This is described in detail in [ObL88].

V Prototyping the Conceptual Schema

V.1 Motivating the Prototyping Approach

It is widely recognized that a suitable integration of the enduser community in the development process is essential for a successful implementation of information systems. However, the typical enduser is not

1 Pre-defined predicates as *insert*, *member*, and *remove* are used, which are defined for example in [ClM87].

able to understand formal specifications like the conceptual schema of INCOME. Therefore a communication gap appears that should be bridged by using suitable development strategies.

The advantages of prototyping in the field of interactive information system design to support communication between all communities involved in the development process are often postulated (cf. [BKM84]). Prototyping supports an early detection of design errors not yet detected by automatic analysis. The enduser's contributions in the design process as a result from working with early available versions of a system usually improve the acceptance of the final implementation. Prototyping facilitates the stepwise adaptation of the specification to changing requirements during the development process. Those changes may arise from external influences concerning the environment of the application system or may be caused by a better understanding of what the final system can do.

On the other hand, it must be noticed that prototyping may also lead to "dirty programming", if it is applied on the basis of fuzzy user concepts or if the system developer lacks any skills necessary for successful prototyping. Moreover, one must point to the effects of conflicts between the interests of different user communities which are dangerous especially for prototyping projects.

Therefore some authors (e.g. [Flo84, Rid84]) suggest the integration of prototyping approaches within appropriate development strategies or life cycle methods. This suggestion has been adapted for INCOME because the formal concepts of Petri Nets provide a solid basis for the use of prototyping techniques.

The INCOME prototyping approach supports prototyping in two different ways: First the conceptual schema is treated as an *operational specification* (cf. [Zav84]) and hence may be executed directly by a suitable interpreter without any compilation and linking. Second the conceptual schema is transformed to an implementation in a selected target environment.

The advantages of using operational specifications are the prevention of inconsistencies between the prototype and the underlying specification and the low time expense for preparing new prototypes after changes of the system specification. While working with this specification only few implementation aspects are considered - the focus is on determination of the conceptual plausibility of the specified system. At this stage of prototyping it is not yet necessary to specify the system's runtime environment.

To prove adequacy of the proposed solution with respect to implementation aspects and to provide a suitable basis for the implementation of the future application system the *transformation of the conceptual schema* to selected target environments is supported. The way this transformation is done, strongly depends on the selected environment and especially on the tools to be used for further system development. If there are powerful tools available for the runtime environment such as program generators or fourth generation languages, the target system will be an interface data structure needed by these tools. If there is only a conventional programming environment available, the conceptual schema will be transformed to a set of almost complete program modules and a database schema.

The important features of the INCOME prototyping approach are the support of both the direct execution of the - possibly incomplete - conceptual schema and the transformation to a suitable implementation. All system aspects of the conceptual schema are made visible by prototyping.

The operational specification is presented in terms of forms which are first automatically generated on the basis of the object structure schema and therefore may be used for the prototyping of this partial schema.

This generation process is further described in the following Section. System behaviour is presented as a series of forms representing the objects flowing through the system. In Section V.3 prototyping of system behaviour is illustrated by an example derived from the case study given in [LNO88].

V.2 PROTOTYPING OF OBJECT STRUCTURES

INCOME already supports prototyping at the early stages of the development process. Usually the system designer decides on the application of prototyping. This decision depends on the application area as well as the user and designer preferences. Usually prototyping becomes possible at the moment when the first complete object structures have been integrated in the conceptual object structure schema. This first prototyping is done without any information about system dynamics.

The aim of object structure prototyping is to prove the plausibility of the already specified object structures and to provide a starting point for the evolution of the object structure schema. Moreover, working with the forms is a good means to teach the enduser to apply database-oriented software systems.

Object structure prototyping proceeds as follows: Based on the conceptual object structure schema a set of subschemas is generated that determines the internal structure of the forms. The external representation of those forms is specified in a second step. The form specification is then interpretatively executed providing the user with the usual operations such as insertion, deletion, update, and retrieval of objects represented by those forms. If the presented forms do not meet the user requirements, the specification will be manipulated possibly resulting in a modified object structure schema.

In the remainder of this Section the form specification technique will be briefly sketched. A characteristic feature of this technique is that the specification consists of several components: a general structure specification, a layout specification for the form, and layout specifications for each of the contained fields. The structure consists of a subschema of the object structure schema and is generated automatically with respect to the inheritance rules defined on the semantic hierarchy object model. In this way the form structure corresponds to the structure of possibly complex objects which are relevant in the application area.

A detailed description of the generation algorithm is given in [Sch89]. Due to space limitations this paper only contains an example of a form structure with sink *delivery information* (cf. Figure 8), i. e. this form can be used to work with objects of type *delivery information*. Now it is assumed that in a certain context of the application system the user is only interested in a few parts of the objects of type *delivery information*. These parts are recognizable by a darkened background. Starting with the complete form structure, a graphical editor supports the interactive projection on the interesting parts of the structure. In the example only object types with elementary value sets have been removed from the form structure. However, removing an object type O from the structure generally causes the removal of that partial structure which contributes to the domain of object type O.

As soon as the relevant form structure is specified, a draft external representation of the form is generated. Figure 9 shows the external representation derived from the form structure of Figure 8. Note that the form is already filled with example values.

Especially in the case of more complex form structures the generated draft representation does not satisfy all of the individual user requirements. Hence a WYSIWYG forms editor is available which supports the interactive modification of the external form representation and ensures consistency between the external representation and the underlying structure of the form.

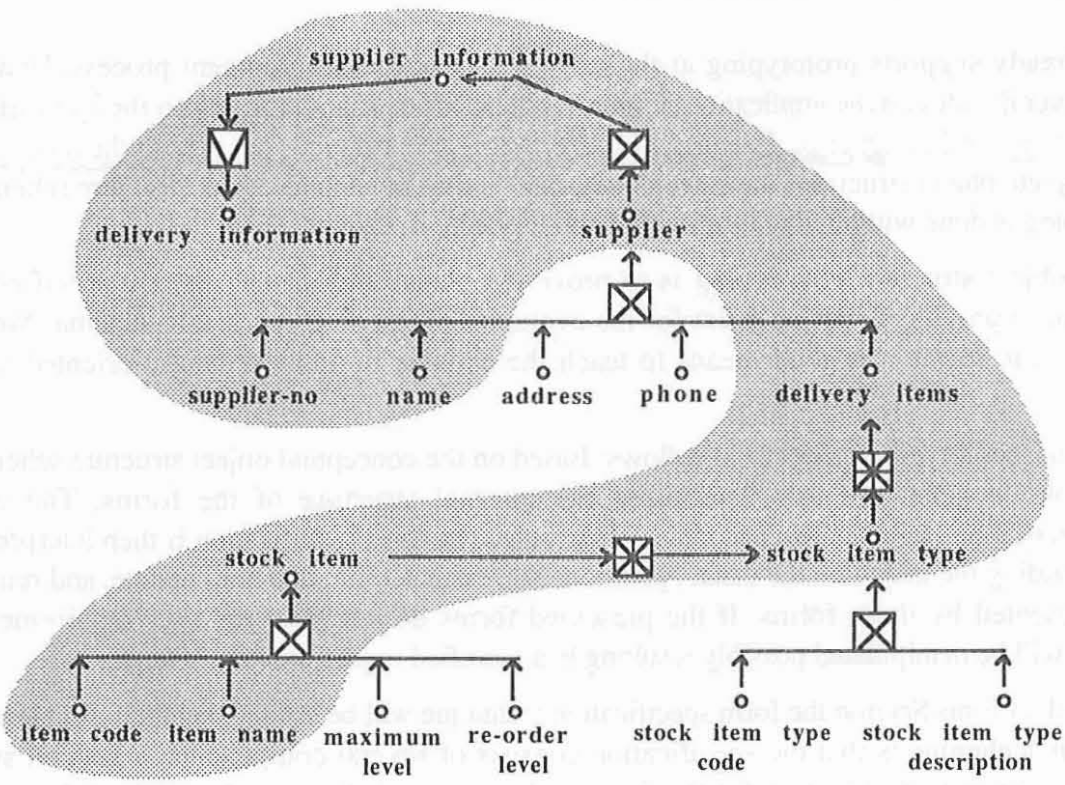


Figure 8: Form Structure with sink *Delivery Information*

Delivery Information

Supplier-No 43 Name Rymans

Code	Stock Item Name
1234	Parker fibre tip refill
3214	Xerox copying paper

Figure 9: Form *Delivery Information*

V.3 PROTOTYPING OF SYSTEM BEHAVIOUR

In the preceeding section we described the procedure of object structure prototyping by means of forms derived from the conceptual object structure schema. However, the essential features of prototyping should be the early availability of an executable system expressing the external appearance of all parts of the conceptual schema. In this way the user of the prototype should be supported in checking the plausibility of all specification aspects and especially the integration of these aspects in the entire schema.

Prototyping the conceptual schema with INCOME means executing the behaviour schema and the integrated transactions with respect to the underlying static specification. Prototype execution is based on the firing of transitions in the behaviour schema which is realized as a PROLOG-inscribed Predicate/Transition-Net. However, the proposed procedure is also applicable for the more simple type of Place/Transition-Nets without any arc or transition inscriptions. In this case the transactions - usually specified by means of transition inscriptions - have to be simulated by user interaction.

Prototype execution is an interactive process, during which the user may ask questions like:

(1) Which transitions are enabled?
(2) Which are the enabling objects?
(3) Which transitions are in conflict with each other?
(4) Which transitions may occur concurrently?
(5) Which transitions may occur sequentially?

Prototype execution starts after initialization of the schema by insertion of objects in some of the places of the behaviour schema. The objects are internally represented as PROLOG data structures; their external representation are forms. The insertion of objects is supported by the forms interface outlined in the previous Section.

For the determination of the enabled transitions the transition formulas have to be evaluated. For this PROLOG programs are generated for each possibly enabled transitions. Each of these programs include the possible input variables and the structures of the output variables as PROLOG clauses as well as the transition formula as a PROLOG rule. These programs are then evaluated by a PROLOG interpreter.

Prototype execution will continue, if the user selects a single enabled transition, a set of such transitions for firing concurrently, or a certain object to be processed. If this selection causes any conflict situation, the conflict will be solved by application of a menu component asking the user for a decision.

The concurrently occurring transitions may be controlled via a multi-window interface. The consumed objects may be inspected using the form interface. If there are any uninstantiated components of those objects, the form interface will also support insertion of values in a way such that the transition formula always holds. Analogously the interactive modification of output objects is supported. These utilities are of great importance to enable prototype execution even at a time when transaction specification has not yet been completed.

Firing a transition will now be further explained in a small example. Figure 10 shows the surrounding of transition *Update Stock Replenishment Data*. Each of the input places contains one object each of them given by its form representation. For the evaluation of the transition formula these objects must be assigned to the input variables A and B.

Figure 11 shows the objects assigned to variables C and D after the evaluation of the transition formula. Note that the components *stock item* and *replenishment* refer to uninstantiated variables and hence have been instantiated by user interaction. The goal *ia_create_stock_replenishment_advice(D)* explicitly specifies this interactive step.

Firing the transition results in a new marking for the behaviour schema. Prototype execution will terminate, if the user aks for termination or if no more enabled transitions can be found.

As also proposed in [WPS86] prototype execution is recorded by means of a so-called *logfile*. This file may be the basis for runtime analysis and several statistics. At any moment the schema can be recovered by markings stored in the logfile. Moreover, former sessions can be replayed by evaluating the logfile.

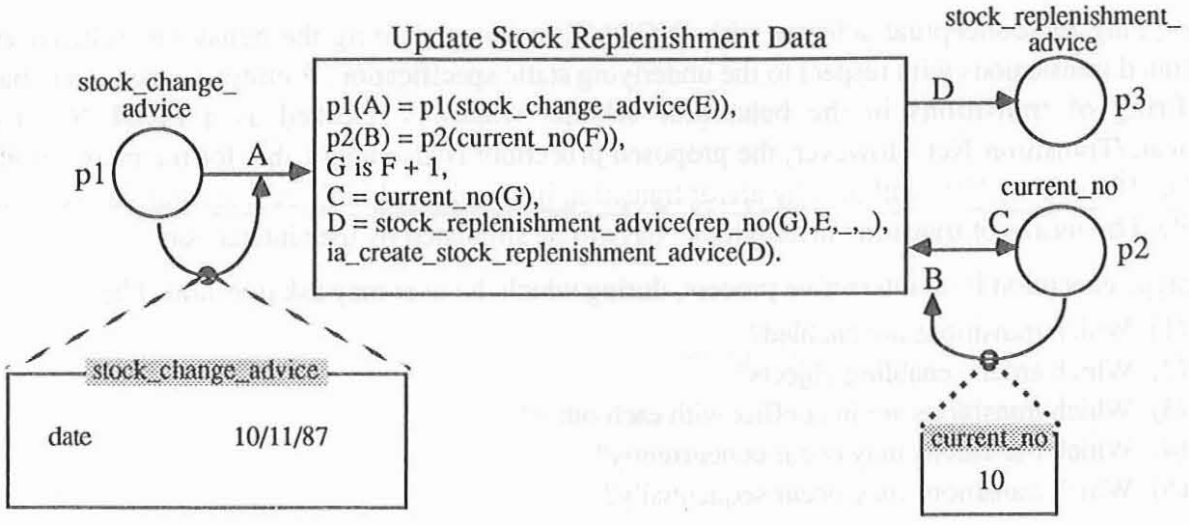


Figure 10: Firing of Transition *Update Stock Replenishment Data*

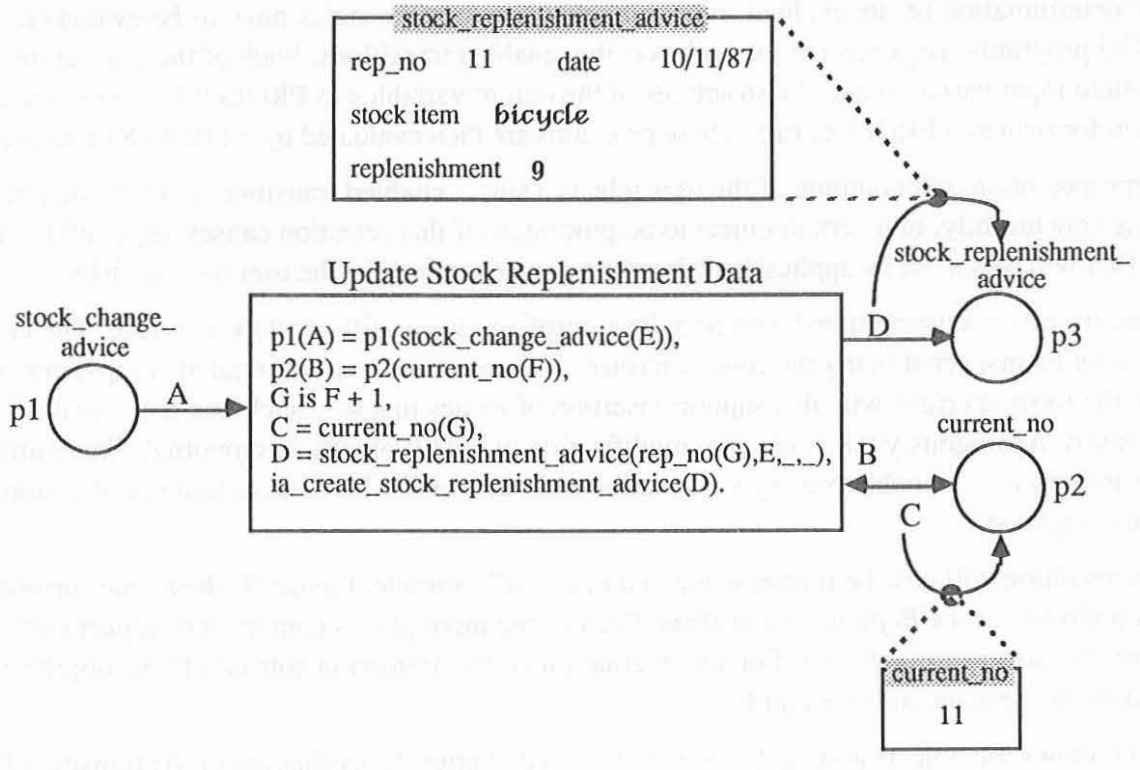


Figure 11: Firing of Transition *Update Stock Replenishment Data*

VI INCOME TOOL SUPPORT

VI.1 ARCHITECTURE OF THE INCOME PROTOTYPE

Starting with the concepts described in the previous sections the prototype of the software development environment INCOME has been implemented on personal computers IBM-AT running the operating system MS-DOS[2]. The program modules are realized in PASCAL. Part of the system runs in a UNIX-based workstation environment (cf. [OSS87]). The future plans are to redesign and reimplement the whole INCOME system to run in a UNIX-based workstation environment.

Figure 12 shows the architecture of the running prototype. The INCOME system consists of three parts: the operating environment, the INCOME toolbox and the development database.

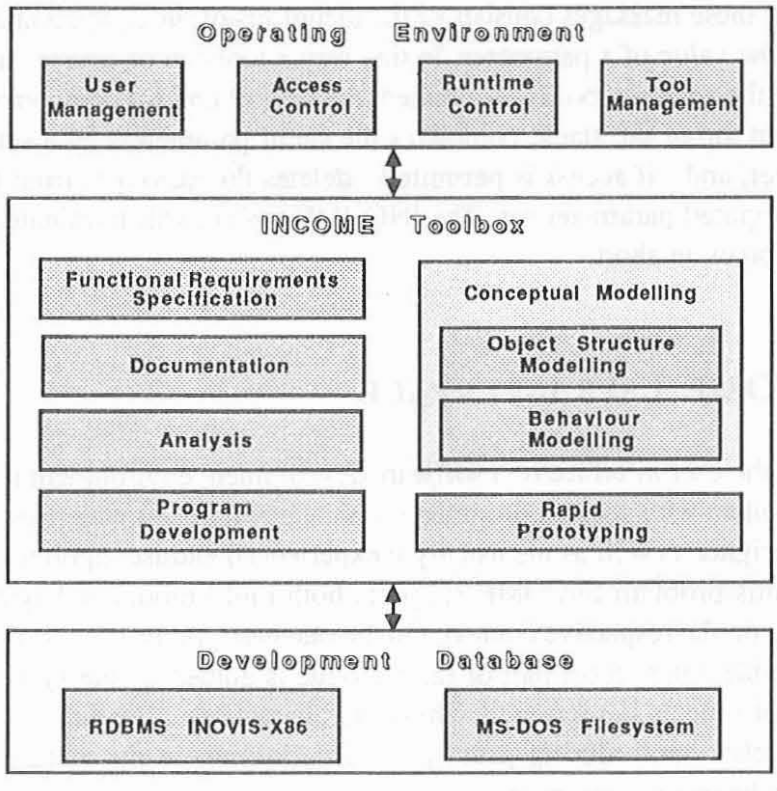


Figure 12: Architecture of the INCOME Prototype

The kernel of the system - the *INCOME toolbox* - consists of several tools supporting the complete software development process based on the INCOME method for conceptual modelling. The toolbox offers tools to support functional requirements specification with object flow diagrams, conceptual modelling including object structure modelling and behaviour modelling as well as documentation and analysis tools. The toolbox also includes a set of tools for the prototyping of the system specification. To support conventional program development tools like editors, compilers, debuggers, a linker and a library manager have been integrated.

To link the tools, INCOME supports the indirect tool communication by providing well defined interfaces to a central development database. The *development database* is managed using the extended relational DBMS INOVIS-X86[3] and the MS-DOS file system. Although the used DBMS is well

2 MS-DOS is a registered trademark of Microsoft Corporation.

3 INOVIS-X86 is a trademark of INOVIS GmbH & Co., Karlsruhe, West-Germany

equipped for the management of development data and does a lot of integrity checking by itself, this is not enough to preserve integrity of the INCOME development database. The main reasons are the necessity of long transaction support and of proving integrity between the relational database and the MS-DOS files. For application in the INCOME environment a concept has been designed based on three mechanisms: data encapsulation by providing predefined operations for database updates, user controlled execution of integrity checking procedures and integration of a knowledge based integrity preserving component (this component called the design expert is still under development).

INCOME is implemented as an open system and therefore supports the augmentation of the toolbox by tools of any kind. The *operating environment* is the component that provides a homogeneous surface for tool applications and makes the elementary tools of the toolbox act somehow like a general macro tool. For this purpose the environment offers utilities for user and tool management and supports a mechanism controlling access between users and tools as well as inter-tool access.

The INCOME runtime control works as follows: Calling a tool means storing messages on top of a system stack. Each of these messages consists of the identifiers of the sending and receiving tool, a time stamp, the type and the value of a parameter. In this way a tool can of course call a series of tools. As soon as execution of the sending tool is terminated, the central control component becomes active and reads the messages on top of the stack, completes the set of parameters by a set of predefined default values for the receiver, and - if access is permitted - deletes the messages from the stack and calls the receiver with the computed parameter set. The INCOME system will terminate, if the system stack is empty or if the user forces an abort.

VI.2 THE INCOME USER INTERFACE

It is well known that the user interface of a software development environment is an essential factor for its valuation. The problem with such environments which are equipped with prototyping facilities is that the skilled system designer as well as the usually inexperienced enduser applies the environment's user interface. To solve this problem INCOME supports both multi-modal and system-directed dialogue techniques depending on the respective context. On the one hand, in the context of conceptual modelling a technique is preferable where most part of the dialogue is guided by the system designer and where direct manipulation of objects is supported. These are characteristic features of multi-modal dialogue techniques. On the other hand, during prototype execution where the enduser is involved, a more restrictive dialogue technique is appropriate.

Figure 13 shows an example of a screen possibly occurring during the execution of a behaviour net with the INCOME prototyping facility. The window and menu techniques supported by INCOME are similar to that of XEROX's STAR (cf. [SIK82]).

The screen is divided up into five parts: the frame with the header on the top and a set of currency indicators at the bottom, the menu bar with the top level functions, the scroll bars, and the working area. The functions of the menu bar refer to the object displayed in the working area. In the example this object is a behaviour net to be executed. More detailed functions are offered via pull-down menus usually displayed after selection of a menu item (in our example the menu item *Tools* has been selected). Depending on the functions that have been selected, further windows are popped-up in the working area. Many functions require the direct selection of parts of the displayed object (sub-objects). Functions to be applied on sub-objects are selected directly via pop-up menus which are displayed near the respective sub-object. This technique speeds up and facilitates working with the INCOME tools.

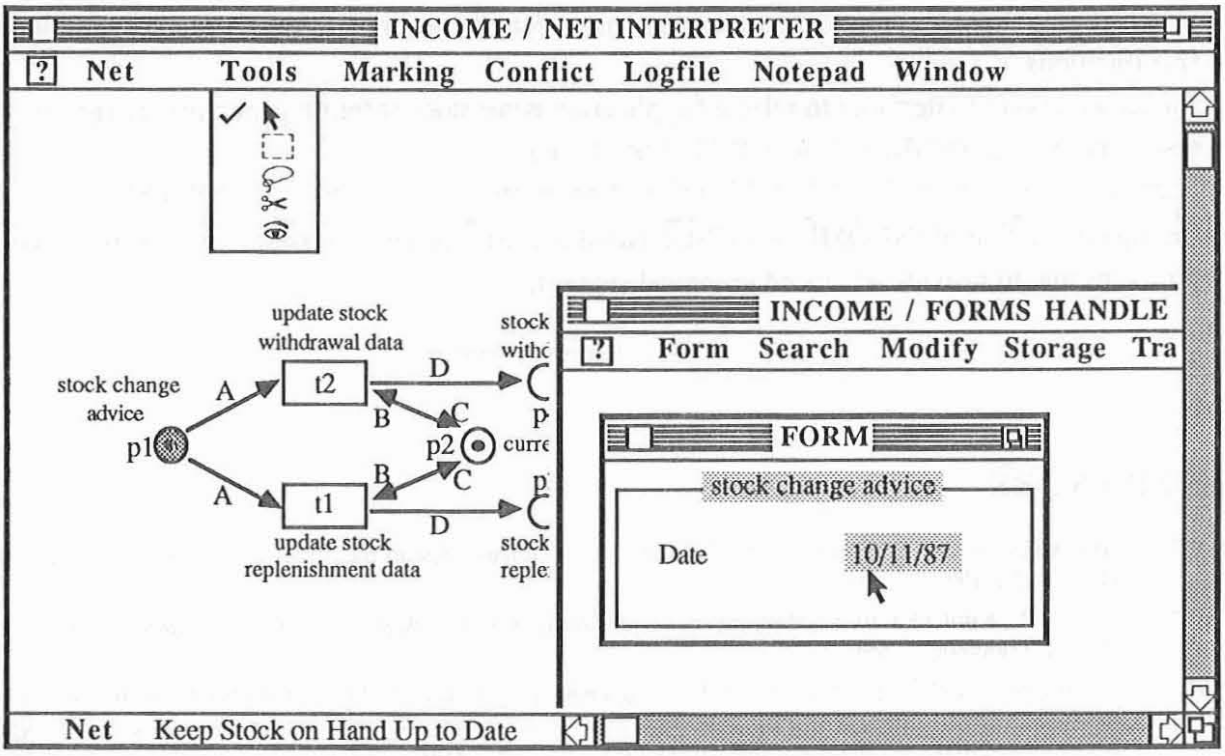


Figure 13: Example Screen

VII Summary and Outlook

In this paper INCOME has been described which is an integrated approach for conceptual modelling and prototyping of information systems. Conceptual modelling depends on a functional requirements specification in a hierarchy of object flow diagrams. INCOME supports the conceptual modelling of object structures and system behaviour on both the user and the database level. The proposed method is constructive in such a way that first draft versions of a new part of the specification are derived from the already present parts of the specification.

Other special features of INCOME are the availability of a uniform formalism for the description of all relevant system aspects and the possibility of working with early prototypes of the future system.

The prototypes provide a powerful forms interface that can be simply adapted to specific requirements of the application. By this prototypes are well suited to be operated by the future enduser of the system. INCOME supports both an interpretative and a transformative prototyping approach. As long as conceptual modelling is still going on, the available prototypes consist of the specification itself and a suitable interpreter. After termination of the specification step, INCOME provides tools for transforming the conceptual schema into an appropriate target system. This system may be a set of program modules together with a database description or an interface data structure to be further processed by using a toolset which is possibly available for the target environment. The transformation step allows to combine INCOME with application generators and fourth generation languages.

Our future plans can be sketched as follows:

- Completion of the INCOME tool set.
- Improvement of the components for conceptual schema analysis.
- Simplification of the handling of the sometimes complex transaction specifications.

- Completion of the interpretative prototyping component by enabling the evaluation of transaction specifications.
- Implementation of interfaces to selected application generators or fourth generation languages respectively (e.g. INGRES, NATURAL, ORACLE).
- Further examination of the INCOME approach in practical case studies (cf. [NSS88]).
- Reimplementation of INCOME in a UNIX-based workstation environment to overcome space problems and to provide advanced graphical support.

REFERENCES

[AnL85] Antonellis, V. De and Leva, A. Di. DATAID-1: A database design methodology. *Information Systems 10*, 2 (1985), 181-195.

[BKM84] Budde, R., Kuhlenkamp, K.,Mathiassen, L., and Züllighoven H. *Approaches to Prototyping*. Springer-Verlag, Berlin, Heidelberg, 1984.

[BrR84] Brodie, M.L. and Ridjanovic, D. On the design and specification of database transactions. In *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases, and Programming Languages*, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, Eds. Springer-Verlag, New York, 1984.

[BRR87] Brauer, W., Reisig, W., and Rozenberg, G., Eds. *Petri Nets: Central Models and Their Properties, LNCS 254*, Springer-Verlag, Berlin, Heidelberg, 1987.

[ClM87] Clocksin, W.F. and Mellish, C.S. *Programming in PROLOG*. Springer-Verlag, Berlin, Heidelberg, 1987.

[Flo84] Floyd, C. A systematic look at prototyping. In *Approaches to Prototyping*, R. Budde, K. Kuhlenkamp, L. Mathiassen, and H. Züllighoven, Eds. Springer-Verlag, Berlin, Heidelberg, 1984.

[HeR86] Heuser, C.A. and Richter, G. On the relationship between conceptual schemata and integrity constraints on databases. In *Database Semantics (DS-1)*, T.B. Steel jr. and R. Meersman, Eds. Elsevier Science Publishers B.V., 1986.

[HNS87] Hohenstein, U., Neugebauer, L., Saake, G., and Ehrich, H.-D. Three-level specification of databases using an extended entity-relationship model. In *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen, Informatik-Fachbericht 143*, R.R. Wagner, R. Traunmüller, and H.C. Mayr, Eds. Springer-Verlag, Berlin, Heidelberg, 1987.

[ISO82] Griethuysen, J.J. Ed. *Concepts and Terminology for the Conceptual Schema and the Information Base*, Report of the ISO/TC97/SC5/WG3, Publ. No. ISO/TC97/SC5-N695, 1982.

[KiM84] King, R. and McLeod, D. A unified model and methodology for conceptual database design. In *On Conceptual Modelling. Perspectives from Artifical Intelligence, Databases, and Programming Languages*, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, Eds. Springer-Verlag, New York, 1984.

[Lau86] Lausen, G. Conceptual modelling based on net refinements. In *Database Semantics (DS-1)*, T.B. Steel jr. and R. Meersman, Eds. Elsevier Science Publishers B.V., 1986.

[Lau88] Lausen, G. Modelling and analysis of the behaviour of information systems. *IEEE Trans. Softw. Eng. 14*, 11 (Nov. 1988), 1610-1620.

[LNO88] Lausen,G., Németh, T., Oberweis. A., Schönthaler, F., and Stucky, W. *The INCOME Approach for Conceptual Modelling and Prototyping of Information Systems*. Forschungsbericht 194, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Univ. Karlsruhe, 1988.

[Lun82] Lundeberg, M. The ISAC approach to specification of information systems. In *Information Systems Design Methodologies: A Comparative Review*, T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, Eds. North-Holland Publ. Comp., Amsterdam, New York, Oxford, 1982.

[NaG82] Navathe, S.B. and Gadgil, S.G. A methodology for view integration in logical database design. In *Proc. of the 8th Int. Conference on Very Large Data Bases*. 1982, pp. 142-164.

[NiV86] Niehuis, S. and Victor, F. *Modellierung und Simulation von Pr/T-Netzen in Prolog*. Arbeitspapiere der GMD 231, Gesellschaft für Mathematik und Datenverarbeitung mbH, St. Augustin, 1986 (in German).

[NSS88] Németh, T., Schönthaler, F., and Stucky, W. Das experimentelle Entwicklungssystem INCOME. In *Anleitung zu einer praxisorientierten Software-Entwicklungsumgebung, Vol. 2*, Th. Gutzwiller and H. Österle, Eds. AIT-Verlag, Hallbergmoos, 1988 (in German).

[Obe88] Oberweis, A. Checking database integrity constraints while simulating information system behaviour. In *Proc. 9th European Workshop on Applications and Theory of Petri Nets* (Venice, Italy, June), 1988.

[ObL88] Oberweis, A. and Lausen, G. On the representation of temporal knowledge in office systems. In *Proc. of the IFIP TC8/WG 8.1 Working Conference Temporal Aspects in Information Systems (TAIS'87)* (Sophia-Antipolis, France), C. Rolland, M. Leonard and F. Bodard, Eds. North-Holland, 1988.

[OSL86] Oberweis, A., Schönthaler, F., Lausen, G., and Stucky, W. Net based conceptual modelling and rapid prototyping with INCOME. In *Proc. of the 3rd Conference on Software Engineering* (Versailles, France, May 27-30). A.F.C.E.T., Paris, 1986, pp. 165-176.

[OSS87] Oberweis, A., Schönthaler, F., Seib, J., and Lausen, G. Database supported analysis tool for Predicate/Transition Nets. *Petri Net Newsletter 28*, (Dec. 1987), 21-23.

[OST83] Olle, T.W., Sol, H.G., and Tully, C.J., Eds. *Information System Design Methodologies: A Feature Analysis.* North-Holland Publ. Comp., Amsterdam, New York, Oxford, 1983.

[OSV82] Olle, T.W., Sol, H.G., and Verrijn-Stuart A.A., Eds. *Information System Design Methodologies: A Comparative Review.* North-Holland Publ. Comp., Amsterdam, New York, Oxford, 1982.

[Ric85] Richter, G. Clocks and their use for time modeling. In *Information Systems: Theoretical and Formal Aspects*, A. Sernadas, J. Bubenko jr., and A. Olivé, Eds. IFIP, 1985

[Rid84] Riddle, W.E. Advancing the state of the art in software system prototyping. In *Approaches to Prototyping*, R. Budde, K. Kuhlenkamp, L. Mathiassen, and H. Züllighoven, Eds. Springer-Verlag, Berlin, Heidelberg, 1984.

[Ros77] Ross, D.T. Structured analysis (SA): a language for communicating ideas. *IEEE Trans. Softw. Eng. 3*, 1 (Jan. 1977), 16-34.

[Sch84] Schiel, U. A semantic data model and its mapping to an internal relational model. In *Databases - Role and Structure*, P.M. Stocker, P.M.D. Gray, and M.P. Atkinson, Eds. Cambridge Univ. Press, Cambridge, 1984.

[Sch89] Schönthaler, F. *Rapid Prototyping zur Unterstützung des konzeptuellen Entwurfs von Informationssystemen.* Dissertation, Univ. Karlsruhe, 1989 (in German).

[SIK82] Smith, D.C., Irby, C., Kimball, R., and Harslem, E. The STAR user interface. In *Proc. of the AFIPS National Computer Conf.*, 1982, pp. 515-528.

[SmS77] Smith, J.M. and Smith, D.C.P. Database abstractions: aggregation and generalization. *ACM Trans. Database Syst. 2*, 2 (1977), 105-133.

[SoK86] Solvberg, A. and Kung, C.H. On structural and behavioural modelling of reality. In *Database Semantics (DS-1)*, T.B. Steel jr and R. Meersman, Eds. Elsevier Science Publishers B.V., 1986.

[SOL87] Schönthaler, F., Oberweis, A., Lausen, G., and Stucky, W. Prototyping zur Unterstützung des konzeptuellen Entwurfs interaktiver Informationssysteme. In *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen, Informatik-Fachbericht 143*, R.R. Wagner, R. Traunmüller, and H.C. Mayr, Eds. Springer-Verlag, Berlin, Heidelberg, 1987 (in German).

[StH86] Studer, R. and Horndasch, A. Modeling static and dynamic aspects of information systems. In *Database Semantics (DS-1)*, T.B. Steel jr., and R. Meersman, Eds. Elsevier Science Publishers B.V., 1986.

[Vos87] Voss, K. Nets in data bases. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, LNCS 255*, W. Brauer, W. Reisig, and G. Rozenberg, Eds. Springer-Verlag, Berlin, Heidelberg, 1987.

[War86] Ward, P.T. The transformation schema: an extension of the data flow diagram to represent control and timing. *IEEE Trans. Softw. Eng. 12*, 2 (Febr. 1986), 198-210.

[WPS86] Wasserman, A.I., Pircher, P.A., and Shewmake, D.T. Building reliable interactive information systems. *IEEE Trans. Softw. Eng. 12*, 1 (Jan. 1986), 147-156.

[Zav84] Zave, P. The operational versus the conventional approach to software development. *Comm. of the ACM 27*, 2, (Febr. 1984), 104-118.