Object Oriented Approach to Software Engineering for CIMIA

Florin-Gheorghe Filip and Gabriel Neagu
Research Institute for Informatics
8-10 Averescu Avenue,
71316 Bucharest
ROMANIA

Dr **Florin-Gheorghe Filip** graduated in Control Engineering and took his Dr Engng. Sc. degree from the Polytechnical Institute of Bucharest in 1970 and 1981, respectively. In December 1991, Dr Filip was elected as a corresponding member of the Romanian Academy. He is a member of the IT Section of the Romanian Academy.
Since May 1991 Dr Filip has been the general director of the Research Institute for Informatics. In November 1991 he was elected as a vice-president of the Romanian Society for Automation and Technical Informatics-SRAIT.
Dr Filip is the author/co-author of some hundred technical papers published in contribution books and international scientific journals such as: IFAC J. Automatica, Large Scale Systems, Computers in Industry, Modelling and Simulation, Systems Analysis, etc. He is co-author of two books:"Cybernetics, Automation and Informatics in the Chemical Industry" (1979), and "Hierarchical Real-Time Systems" (1986) . Dr Filip was session chairman and/or IPC member at several international scientific conferences such as IEEE International Conference on Systems, Man and Cybernetics (1988), Systems Analysis and Simulation (1988), IFAC/ IFORS Symposium on Large Scale Systems(1992), etc. He was invited to deliver lectures and participate in conferences in China, Czech Republic, England, France, Germany, Kuwait, Poland and Sweden. He is the editor-in-chief of the " Studies in Informatics and Control" Journal. Dr Filip' s main current scientific interests include hierarchical optimization and control of large scale systems, decision support systems, integrated plant control in the process industries and discrete part manufacturing and information systems in public administration.

Gabriel Neagu was born in Bucharest in 1949. In 1973 he received his MSc. in Control Engineering from the Energetic Institute of Moscow, and joined the Research Institute for Informatics in Bucharest. His major Research and Development activities have been related to complex information systems for steel works and aluminium plants, pipe factories, electronic industries, real-time discrete production control systems at the shopfloor level, MIS at micro and macroeconomic level. He is author/ co-author of more than 20 papers, mainly dedicated to industrial information systems design. Currently, he is senior researcher at the Research Institute for Informatics. He is working towards obtaining a doctoral degree in applied informatics. His research interests include knowledge-based production control systems, modelling and qualitative simulation of discrete processes, decision support systems, societal informatization, advanced informatic systems engineering.

1. Introduction

The object-oriented approach (OOA) is one of the most actively addressed topics in today software engineering. The main advantages of this approach include structuring capabilities, unity of the communication mechanism between modules, diminution of code redundancy, clear delimitation of programmers' tasks, increasing reusability of previously generated code.

For system analysis and design activities the major benefit of OOA consists in the natural matching between reality and model. Abstraction power, structuring flexibility, interaction expressiveness, robustness of OO design with respect to the requirement changes, are but some most attractive OOA features for these activities. Seven prime motivations and benefits of the OO analysis are identified in [C&Y91a]: tackle more challenging problem domains, improve analyst and problem domain expert interaction, increase the internal consistency of analysis results, explicitly represent commonality, build specifications resilient to change, reuse analysis results, provide a consistent underlying representation.

There are reported a lot of attempts at and some significant results in using OOA to treat the specificity and complexity of the Computer Integrated Manufacturing and Intelligent Automation (CIMIA) field. The paper aims at providing a general view on OOA subject and its utilization in CIM environment. As far as a specific application domain is concerned, the emphasis is put on the OO analysis and design methods (OOADM). Section 2 presents the basic

concepts of OOA, and also the evolution of the major OO information technology segments: programming language, databases, OOADM. A short characterization of the specificity of the OO life-cycle is also provided. Section 3 is devoted to a short presentation of the most representative purely general-purpose OOADM. The intention is twofold: to give an image of the extent to which these methods cover system development requirements in the CIMIA field, and to suggest the necessity for further efforts towards a unified approach of the development process structure and the semantics of representations. Major orientations in specific OOADM development for CIM environment are outlined in Section 4. Some conclusions and proposals for prospective efforts in this field are formulated in the final section.

2. Object Oriented Approach

2.1 Basic Concepts

Every *object* encapsulates a state (a set of values for its attributes) and a behaviour (a set of methods that operate on its state).

All objects that share the same set of attributes and methods are gathered as a *class*, such that an object belongs to only one class as an *instance* of that class. Class is a generic concept, a model of an object. Further, a class may be considered as an object and then its model is a *metaclass*.

All classes are organized as a rooted, directed acyclic graph or as a hierarchy (called *class hierarchy*). A class *inherits* all the attributes and methods from its direct and indirect ancestors in the class hierarchy. Semantically, a class is a specialization (subclass) of the class(es) which it inherits attributes and methods from; the same class is a generalization (superclass) of the classes that inherit attributes and methods from it. The class hierarchy must be dynamically extensible, i.e. a new subclass can be derived from one or more existing classes. The case when the superclasses of a class are not interconnected in the class hierarchy, corresponds to the *multiple inheritance*.

The domain of an attribute of a class may be any class. This leads to a directed graph of classes called *aggregation hierarchy*, which may be cyclic: the value of an attribute of an object is also an object, there resulting a *composite object*.

The state and behaviour encapsulated in an object may be accessed or invoked from outside the object only through explicit *message passing*. Further, a message sent to an instance of a class may be bound to a method defined in a superclass of the class. The object *interface* is defined by the set of messages meaningful for that object. According to the *polymorphism* concept, different objects may react different ways to the same message, thus allowing the simplification of the interface between objects.

2.2 OO Information Technology Development

Programming Languages. Perhaps every history about OO language (OOL) should start with Simula language [D&N66], which introduced the term of *encapsulation* and the concept of *virtual machine* with data and operations working together. After that the next significant step was represented by Smalltalk language, considered to be the first OOL [G&R83]. Later, by mid'80ies many other OOLs emerged, asking for a lot of attention and efforts to compare and systematize them. In [Tel89] OOLs are classified into: (a) *exclusive (pure) OOL*, build around the concept of object (like Smalltalk), and (b) *hybrid OOL* as, for example, the extensions of conventional programming languages (Flavors [Moo86], CLOS, Object Lisp - for Lisp, C++ [Str87], Objective C [Cox86] - for C, Pascal-Object [Int89] - for Pascal). Based on the distinction between the concepts of class and instance, [Arn] proposes 3 groups of OOL, respectively based on: (a) objects belonging to classes (the most representative group including both exclusive and hybrid OOL mentioned above), (b) generalized objects, where class and instance are less exclusive (for example knowledge representation languages like KEE, with a unique basic structure for all objects), and (c) active objects (actors), with a highly dynamic evolution and behavioural autonomy.

Different criteria to define a true OOL and to discourage unreasonable attempts at penetrating

this area have also been proposed. Thus, Meyer [Mey88] formulates 7 successive levels of object orientation which should be implemented by an OOL: object-based modular structure, data abstraction, automated memory management, classes (as a combination of module and type), inheritance, polymorphism, and multiple inheritance. Also [Arn90] defines three properties common to all OOLs: use of object structure; share the object knowledge by either inheriting its information or using the object in different procedures, and provide the object with the message mechanism for communication with its environment.

Databases. The field of OODBS has started to spread out significantly by mid'80ies. In [Kim91] there are identified two major reasons why using the OO paradigm as a basis for a new generation of database technology: (1) a data model that subsumes the data models of conventional database systems can be constructed using this paradigm, and (2) the notions of encapsulation and inheritance (reuse) impair the difficulty of developing and evolving complex software systems. It should be reminded that the same goal of overcoming the difficulties has driven data management technology from file systems to relational database systems during the past three decades.

According to [C&F92], an OODBS integrates features from relational databases (storage management methods such as persistence, transaction, concurrency, recovery, querying, versioning, integrity, security), from semantic data model (aggregation, generalization, and specialization associations), and from OO programming (polymorphism, classes, methods, encapsulation, reusability and extensibility).

Relevant results in the field appeared at the beginning of the 90ies: Gemstone [BOS91], Ontos [HAD90], Orion [Ki&90], O2 [De&91].

Some weaknesses of OODBS are formulated in [Kim91]: commercial offerings suffer in varying degrees in performance and/or functionality, the richness of an OO data model also implies extra difficulties in implementing an OODBS that will render high performance, the lack of an industry-wide consensus on the semantics of the OO paradigm beyond a set of high-level OO concepts.

Analysis and Design Methods. The major advantage of OOA for system analysis and design activities is the natural matching between the reality and the model. That explains the interest of this approach in simulation, where Simula has been considered as a precursor of Smalltalk. Abstraction power, structuring flexibility, interaction expressivity are but some most attractive OOA features for these activities.

The Booch's paper [Boo86] published in 1986, is considered to be one of the first sound bases in this field. Starting with the end of the 80ies a lot of attempts at developing analysis, modelling and design methods, were reported. Some of the most representative ones will be described in more detail in the next section in order to emphasize the importance of this OO IT component. Now some general remarks on this topic are made in the sequel.

First, the favourable factor of a demanding context seems to have played an important role as to increasing the interest in this field. Thus, one source of the software crisis that characterized the first half of the '80ies was the lack of *requirement analysis* methods adapted to the software engineering approaches. The OOA was identified as one of the most promising solutions in filling this gap. Also, in the mid80ies, the *prototyping approach* started to play an increasing role in software engineering as a solution for an efficient support of the end-user's involvement in early phases of the life cycle [Flo84]. OOA is considered to be very suitable for this goal due the natural way of communication between end-user and development team that it hosts.

Second, the IT community are ever more aware of the potential of these methods, as emphasized by different comparative analyses in this field. Coad and Yourdon provide such a comparison [C&Y91], which includes functional decomposition, data flow, information modelling and OO approaches. The comparison is guided by the so-called principles of managing

complexity: abstraction, encapsulation, inheritance, association, communication with messages, categories of behaviour.

Third, as a consequence of the increasing importance attached to requirements definition phase, the emphasis in OO methods development is put on the *analysis methods*.

According to [Br&91] there are 3 major orientations in this area: (a) methods based on *entity-relationship model*, where the key problem consists in finding out the solution capable of treating the relation as class; (b) methods based on *operational approach* working with structural view (objects, structures, relations) and behavioural view (states, events, methods, messages) of the system; and (c) methods based on the *declarative approach*, treating the system as a collection of interrelated objects.

Another significant trend in this field aims at reconciling structured analysis, based on the data flow, with OO approach. An example is given in [War89], where the data flow derived from the functional decomposition is replaced by the "stimulus-answer" type analysis.

2.3 OOA Life Cycle

The modelling power and expressivity of the object oriented paradigm encourage prototyping techniques. This leads to an *incremental model* of system development life cycle, with its well-known advantages: useful and meaningful feedback to the user, major system interfaces tested first and most often, a smooth transition between system versions, overlapping analysis, design and development phases to provide early results. The life cycle for OO design proposed in [Boo91] includes the following phases: *analysis, design, evolution* and *modification*. The author is utterly critical to separating the activities of analysis and design, the proposed strategy being "analyze a little, design a little". The evolution phase combines traditional coding, testing and integration aspects. Prototyping is strongly encouraged in this phase as well. Adding a new class, changing the implementation of a class, re-organizing the class structure, changing the interface of a class, are types of evolutionary changes. The modification phase consists in adding new functionality or in modifying some existing behaviour.

As a general remark, in such incremental type life cycle the mainstream of the system development process has further a downward orientation, even if there are significant upward feedback links between phases. In our opinion, the efforts in the life cycle model development for OOA should be directed towards the idea of *reusability*, which is a key advantage of the object-oriented paradigm.

As far as a concrete application domain, like CIMIA, is concerned, besides the foreground process of the *incremental development* of a new application, an upward, background process of *cumulative development* of the domain class hierarchy design and implementation have to be considered. The final goal of the background process should be a computer-implemented reference model. A *two-phase computer-aided development process* of a new application will then be available: (1) *a specialization phase* standing for the requirement definition as a subset of existing reference models, (2) *an instantiation phase* of implementation, based on concrete values for system attributes and methods.

This perspective addresses in fact a new significant stage in the development of today *generic modelling approach* to CIM environment [F&N93].

3. General -Purpose OOADM

3.1 Object Oriented Analysis and Design (OOAD) [C&Y91a, b]

The strong point of the method is the analysis phase, which establishes a continuum of representation for systematically expanding its results onto a specific design.

The analysis process is structured in five layers (subject, class & object, structure, attribute and service) and in five corresponding activities:

a. *Finding class & objects*: during this activity structures, devices, things and events, roles played, operational procedures, sites, and

organizational units are analysed. The following criteria should be met: needed remembrance, needed behaviour, (usually) multiple attributes, (usually) more than one object in a class, always applicable attributes and not merely derived results.

b. *Identifying structures*: generalization-specialization structure and whole-part structure with relevance to the given application domain, are defined.

c. *Identifying subjects*, as a basis for the structure definition of the future system model. Principles of minimal interdependence between class structures and of minimal interrelation (message transfer) between objects belonging to different subjects ,are considered.

d. *Defining attributes*. Relevant information about the object features and states in the context of system's responsibilities, is analysed. Identified attributes should be placed as high as possible in the generalization-specialization structure. Instance connections, i.e. links between objects required to fulfill their responsibilities, are also defined.

e. *Defining services*. Simple (implicit) and complex services are identified as a support for the specific behaviour which an object is responsible for exhibiting. This activity also includes the identification of the message connections between objects in order to model their processing dependency.

The design process consists in improving OOA results for designing the problem domain components, and designing human interaction, task management and data management components.

The method underlines the importance of CASE tools for both the analysis and design processes, as a support for prototype development.

3.2 Object Modelling Technique (OMT) [Ru&91]

OMT is a modelling and design method for software development, based on modelling objects from the real world and using the model to build an OO language - independent design.

The software development life cycle is structured in analysis, system design and object design. The method also describes the target environments, including OO languages, non OO languages, and relational databases.

The major concerns of the *analysis phase* are object modelling, dynamic modelling and functional modelling. The *object model* describes the structure of the objects in a system - their identity, their relationship to other objects, attributes and operations (methods). The goal in object modelling is to capture those concepts in the real world that are important for the application. The object model provides the essential framework within which the dynamic and functional models can be placed. The *dynamic model* is the event model of the system, describing aspects of the system concerned with time and sequencing of operations. The dynamic model captures control aspects and is graphically represented by state diagrams. The *functional model* describes what the system does, but not how it does . The model is represented by data flow diagrams.

The operations in the object model correspond to events in dynamic model and functions in functional model. Functions are invoked by operations in the object model and actions in the dynamic model. They operate on data values specified by the object model.

During the *system design phase*, the target system is organized into subsystems based on both the analysis technique and the proposed overall architecture. The system designer must decide on what performance characteristics to optimize, choose a strategy of attacking the problem and make tentative resource allocation.

In the *object design phase* details are added to the design model in accordance with the strategy established during system design, with emphasis on data structures and algorithms needed to implement each class. Computer-domain objects are included in the model. They are described using the same object-oriented concepts and notation as application-domain objects do, even though they exist at different conceptual levels.

In the *implementation phase* the object classes and relationships developed during object design

are finally translated into a particular programming language, database or hardware implementation. The target language influences on design decisions should be minor.

3.3 Object Oriented Development (OOD) [Boo91]

The OOD method is dedicated to complex systems development based on the object model. OOD operates with 5 types of diagrams as a support for the design process:

(1) *Class diagrams*: classes and their relationship in the logical design of a system. Depending on its complexity, a system requires one or more such diagrams to document its class structure.

(2) *State transition diagrams*: dynamic behaviour associated with certain classes through the state space definition, events that cause a transition from one state to another, actions resulting from a state change.

(3) *Object diagrams*: the existence of objects (object visibility) and their relationship (message synchronization) in the logical design of a system. They capture the dynamic semantics of the object operations. Each object in an object diagram denotes some (specific or arbitrary) instance of a class in the class diagram.

(4) *Timing diagrams*: the dynamics of message passing in an object diagram. They provide additional information to the static image of cooperative collection of objects passing messages to one another provided by the object diagrams. Also state transition diagrams only show how state changes occur within a single object, not within a set of collaborating objects.

(5) *Module diagrams*: the allocation of classes and objects to modules in the physical design of a system. A single module diagram represents full or part module architecture of a system.

(6) *Process diagrams*: allocation of processes to processors in the physical design of a complex system designed for a distributed multiprocessor architecture.

The OO design process is considered to be an incremental process which generally tracks the following order of events:

a. *Identify classes and objects of a given level of abstraction*: discover the key abstractions in the problem space (the significant classes and objects) and invent the important mechanisms that provide the behaviour required by objects working together to carry out some functions.

b. *Identify the semantics of classes and objects*: establish the meaning of classes and objects from the perspective of their interface in order to identify things that can be done on each instance of a class and that each object can do to another object. This phase is considered to be much more difficult than the first one.

c. *Identify relationships among classes and objects*: define the static and dynamic semantics of each interaction mechanism in order to ensure the required object visibility.

At this point in design focus is still entirely on the outside view of the key abstraction and mechanisms.

d. *Implement classes and objects*: make design decisions on the representation of classes and objects, and allocate classes and objects to modules (and programs to processors).

4. Specific OOADM for CIM Environment

The support for the OOA implementation in CIM environment is given by both general -purpose and specifically developed OOADM.

In [Ba&93], an OMT model for system dynamic representation is described. The system model is organised in topological, operational and graphical sub-models, describing how the components are interconnected, the function of each component and the graphical representation of each component, respectively.

As to specific development methods, it should be noted that they are usually not so rigorous in satisfying the OOA requirements. It means that the analysis phase is mainly based on traditional methods (functional decomposition, structural analysis). This comes from the efforts of converting the existing experience in using the

traditional methods to this new paradigm. From this point of view specific methods may be considered more pragmatic in this period, by smoothing the process of conscious transition towards a new development framework. Obviously this class of methods corresponds to the trend of reconciliation mentioned at Section 2 in the context of OOADM taxonomy definition.

HOOD (Hierarchical Object Oriented Design) method [HOO89] is destinated to the development of complex real-time software systems and has been developed for the European Space Agency.

Each object is defined by its *static properties* (object body, required interface, offered interface) and *dynamic properties* (sequential or parallel regime for the required operation execution). There are defined two types of objects: *passive objects* (offering operations running in sequential regime) and *active objects* (with their own internal control structure for operation required by other different objects and running in parallel regime). The objects are related through two kinds of connections: utilisation (an object uses the operations with other objects) and integration (an object integrates in its structure other objects). Concerning the development life cycle, HOOD method is used for the architectural and detailed design phases.

The requirement analysis phase is supposed to be carried out with other both traditional (SADT) and object oriented (OOAD, OOSA) methods. At the output, like other OO methods, HOOD provides language-independent specifications, which may be codes in ADA, Pascal, C, OOLs.

A development of HOOD method is presented in [P&R93]: the HOOD/PNO method uses Petri nets to describe the behaviour of active objects.

Another example is provided in [KKC93], where the *OOMIS (Object Oriented modelling methodology for Manufacturing Information Systems)* is presented. In the analysis phase, manufacturing functions are decomposed into component functions using functional diagrams similar to IDEFO. Data flow among functions and also their infrastructure, are defined. Then, functional diagrams are transformed into function, data and operation tables. In the design phase, these tables are translated into an object-oriented information model including the class dictionary (function classes and entity classes) and the class relationship diagrams (generalization, aggregation, and interaction).

There are two other orientations in specific OOADM development. The first one deals with the release of OO versions for former developed methods. An example is the *M* - OBJECT methodology* [DGV], aiming at providing engineers with methods and tools for information system analysis and design as well as the development of database applications in production and CIM environments. It is the OO extension of the *M** methodology based on an extended version of the entity-relationship approach.

This orientation is entitled for the case of general-purpose methods . The Object- Oriented System Analysis [S&M88] is considered as a structured analysis method converted to the OO way of thinking.

The second orientation is less important in the context of this analysis. It includes traditional methods using OO terminology for better formalization of modelling requirements ([N&G94], [Ina93]).

In the OO, analysis and experience coming from the utilisation of OODBS are regarded, and an example is given in [C&F92], where the Orion system is used to develop a BOM system.

In our institute, the OOA is used in the area of DSS for both process and manufacturing industries. Making use of OO concepts, a *multilevel modelling scheme*, including an external model (EM), a mathematical formulation (MF), and an internal (performance) representation (IR), has been developed for the process industry [Fil93]. The OO view of the scheduling problem is presented in Figure 1.

In manufacturing industry, the *(MC-D)S project* aims at providing solutions for Decision Support problems (scheduling, real -time allocation,and monitoring) in Manufacturing shop Control Systems. The project is being developed according to the *generic prototyping approach*

(GPA) [Nea93]. The major product of the requirement definition phase is the *OO conceptual model* of the system including the structural model (Figure 2), the control (behavioural) model, and the user-interface model. The domain specific decisional resources are identified too. For the design specification phase, the system structure is defined using the *GPA hierarchical framework architecture*, with resource, services and option levels. Then the detailed OO specifications are developed and verified through the mechanism of particular prototypes implementation. The generic prototype has gradually been developed by the integration of adopted particular prototypes. The generic prototype for the flow shop model has lately been developed.

5. Conclusions

The attention paid to OOA implementation in the field of CIMIA is to be remarked. Its attractivity is mainly motivated by the *modelling power* of object oriented paradigm, which is generated by both the direct correspondence of the modelling concepts with the real world and the unity of these concepts through the entire development life cycle. This is the reason why most of the efforts and results are reported in the area of system analysis and design.

The *development strategy* is based on either general-purpose or specific methods. The former ones, beyond their practical use, are also a good support to familiarizing system developers with the specificity of this approach. The latter ones seem to be more pragmatical as trying to reconcile the OOA requirements with their authors' former experience. Using the structured analysis as a front-end to the OO design is but one significant example in this respect. Some of the specific methods claiming the object orientation, are in fact traditionally shaped, but using OOA terminology. As the case was with OO programming languages, clear *criteria* should be formulated to discriminate real OOAD methods. More attention and efforts should be devoted *to compatibilise and standardise* the development process structure and graphical notations of today methods. In this respect the large experience in OO software development stimulated by OO technological support available on the market (OODBS, OO programming environments) has to be considered as well.

All analysis and design methods are unanimous in emphasizing the role of prototyping approach. This results in an incremental type life cycle for the system development process. The major beneficial aspect is again the *uniqueness of modelling concepts*. This ensures the compatibility of prototypes developed in different phases of the life cycle and encapsulates the system image with different levels of resolution. The effective exploitation of this advantage requires the development of powerful prototyping tools.

There is another reason why claiming for such support: *the reusability* of former solutions and results. Although it is one of the hit ideas of OO paradigm, the reusability has not been openly addressed until now. With respect to a given application domain, this aspect becomes crucial. Here the OO prototype should evolve gradually towards an implementation-independent generic model for the given domain. Such a prototype, together with a set of compatible class libraries, developed for different computing platforms and configurations, could be viewed as one of the most reasonable goals of the endeavours to fully utilise the OOA capabilities in domain-oriented software engineering.

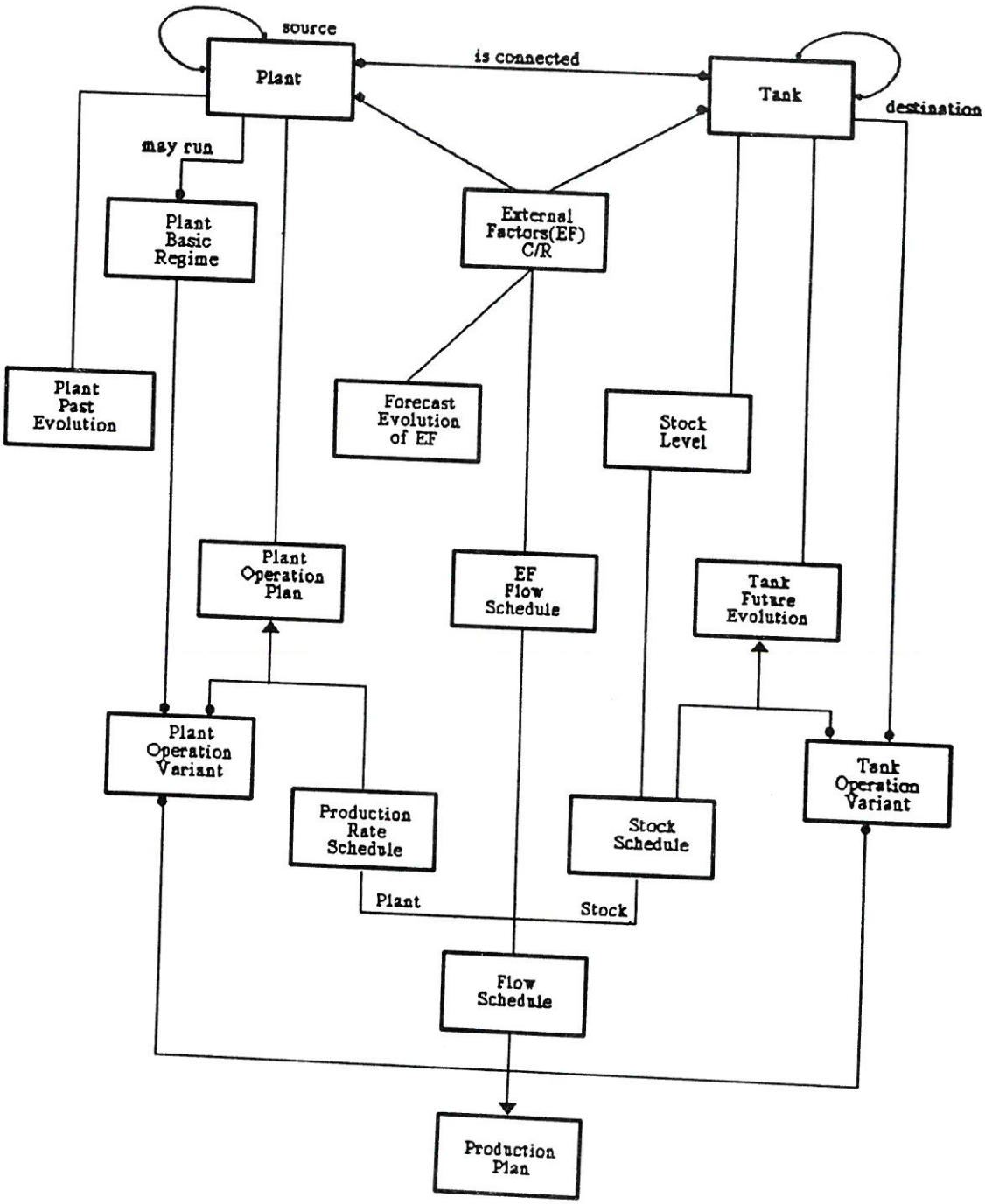

Figure 1. An Object Oriented View of the Scheduling Problem

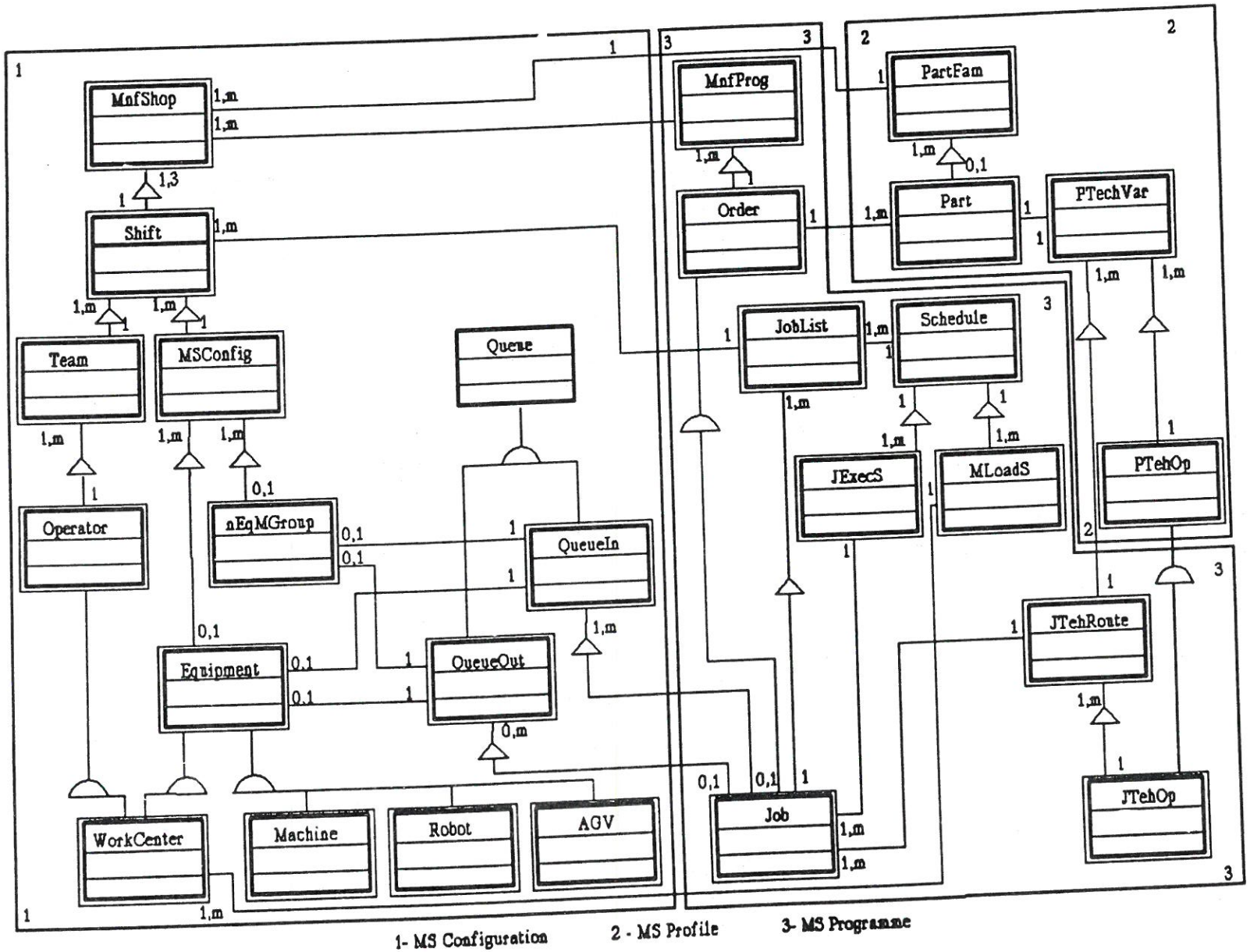

Figure 2. The Object Oriented Model of a Manufacturing Shop

REFERENCES

[Arn90] ARNOUX, M., **Programmation orientée objet et systèmes multi-agents: application en robotique et productique,** Ph. D. Thesis, University of Nice and Sophia-Antipolis,1990.

[Ba&93] BARKER, H.A., HARVEY, I.T., GRANT, P.W. and JOBLING, C.P., **Object-Oriented Data Representation for Computer Aided Control Engineering,** Preprints, IFAC 12th World Congress, Vol. 3, Sydney, 1993, pp. 281-284.

[Boo91] BOOCH, G., **Object-Oriented Design with Applications,** BENJAMIN/ CUMMINGS, Redwood, 1991.

[Boo86] BOOCH, G., **Object-Oriented Development,** IEEE TRANS. ON SOFTWARE ENGINEERING, Vol. 12, No. 2, 1986, pp. 211-221.

[Br&91] BRUNET, J., GROSZ, G., ROLLAND, C., SCHMITT, J.R. and SOUVENET, C., **Information System Development: A Survey,** MASI Report 91.38, Inst. B. Pascal, Paris, 1991.

[BOS91] BUTTERWORTH, P., OTIS, A. and STEIN, J. , **The Gemstone Object Database Management System,** COMMUNICATIONS OF THE ACM, Vol. 34, No. 10, 1991, pp. 64-77.

[C&F92] CHUNG, Y. and FISCHER, G.W. , **Illustration of Object-Oriented Databases for the Structure of A Bill of Materials,** COMPUTERS IN INDUSTRY, Vol. 19, 1992, pp. 257-270.

[C&Y91a] COAD, P.E. and YOURDON, E. , **Object-Oriented Analysis,** YOURDON PRESS, Englewood Cliffs, NJ, 1991.

[C&Y91b] COAD, P.E. and YOURDON, E., **Object-Oriented Design,** YOURDON PRESS, Englewood Cliffs, NJ, 1991.

[Cox86] COX, B.J., **Object-Oriented Programming: An Evolutionary Approach,** ADDISON-WESLEY, Reading, MA, 1986.

[D&N66] DAHL, O.J. and NYGAARD, K., **SIMULA-An ALGOL-Based Simulation Language,** COMMUNICATIONS OF THE ACM, Vol. 9, No. 9, 1966, pp. 671-678.

[De&91] DENK, O. et al, **The O2 System,** COMMUNICATIONS OF THE ACM, Vol. 34, No. 10, 1991, pp. 34-48.

[DGV93] DiLEVA, A., GIOLITO, P. and VERNADAT, F., **The M* - Object Methodology for Information System Design in CIM Environment: the Organisation Analysis Phase,** Research Report No. 1918, INRIA, Rocquencourt, 1993.

[D&V87] DiLEVA, A. and VERNADAT, F., **Information System Analysis and Conceptual Database Design in Production Environments with M*,** COMPUTERS IN INDUSTRY, Vol. 9, 1987, pp.193-217.

[Fil93] FILIP, F.G., **An Object-Oriented Multilayer Model for Scheduling,** in *A.* Verbrack, E.J. Kerckhoffs (Eds.) Proceedings of European Simulation Symposium, Delft, 1993, pp. 173-180.

[F&N93] FILIP, F.G. and NEAGU, G., **CIM in Continuous and Discrete Manufacturing,** CONTROL ENGINEERING PRACTICE, Vol. 1, No. 5, 1993, pp. 815-825.

[Flo84] FLOYD, C. , **A Systematic Look At Prototyping,** in R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Zullighoven (Eds.) Approaches to Prototyping, SPRINGER-VERLAG, Berlin, 1984.

[G&R83] GOLDBERG, A. and ROBSON, D. , **Smalltalk-80; The Language and Its Implementation,** ADDISON-WESLEY, Reading, MA, 1983.

[HAD90] HARRIS, C., ANDREWS, T. and DUHL, J., **The Ontos Object Database Technical Report,** Ontologic Inc., Burlington, MA, 1990.

[HOO89] **HOOD User Manual.** E.S.A., Version 3.0, September 1989, Ref. WME/89-353JB.

[Ina93] INAMOTO, A. , **A Study on Object-Oriented System Design of Factory Automation Systems,** Preprints of IFAC 12th World Congress, Vol. 4, Sydney, 1993, pp. 4-12.

[Int89] INTERSIMONE, D., **Turbo Pascal 5.5, Design Goals and Language Implementation**, Proc. of TOOLS'89 Conf., Paris, 1989, pp. 283-294.

[Kim91] KIM, W. , **Object-Oriented Database Systems: Strengths and Weaknesses,** JOOP, July/August 1991.

[Ki&90] KIM, W., GARZA, J.F., BALLOU, N. and WOELK, D., **Architecture of the ORION Next Generation Database System**, IEEE TRANS. KNOWLEDGE DATA ENG., Vol. 2, No. 1, 1990, pp. 109-124.

[KKC93] KIM, C., KIM, K. and CHOI, I., **An Object-Oriented Information Modelling Methodology for Manufacturing Information Systems**, COMPUTERS AND INDUSTRIAL ENGINEERING, Vol. 24, No. 3, 1993, pp. 337-353.

[Mey88] MEYER, B., **Object-Oriented Software Construction**, PRENTICE-HALL, 1988.

[Moc92] MOCHEL, T., **Simulation of Discrete Systems with An Object -Oriented Concept**, in A. Sydow (Ed.) Computational System Analysis, Proc. of 4th Int. Symposium on Systems Analysis and Simulation, ELSEVIER, Berlin, 1992, pp. 611-616.

[Moo86] MOON, D.A., **Object-Oriented Programming with Flavours,** Proc. of OOPSALA Conf., Special Issue of SIGPLAN Notices, Vol. 21, No.11, ACM, New York, 1986, pp. 1-8.

[Nea93] NEAGU, G., **Generic Modelling Vs. Prototyping: An Object- Oriented Approach to the Decision Support at the Shopfloor Level**, in R.S. Sodhi (Ed.) Proc. of 9th Conf. on CARs&FOF, Newark, NJ, 1993.

[N&G94] NGWENYAMA, O.K. and GRANT, D.A., **Enterprise Modelling for CIM Information System Architecture: An Object- Oriented Approach**, COMPUTERS IND. ENGNG., Vol. 26, No. 2, 1994, pp. 279-293.

[P&R93] PALUDETTO, M. and REYMOND, S., **A Methodology Based on Objects and Petri Nets for Development of Real-Time Software**, Proc. of Int. Conf. SMC'93, Le Touquet, France, 1993.

[Ru&91] RUMBAUGH, J., BLAHA, M., PREMERHANI, *N.*, EDDY, F. and LORENSEN, W., **Object-Oriented Modelling and Design**, PRENTICE-HALL, 1991.

[S&M88] SHLAER, S. and MELLOR, S.J., **An Object-Oriented System Analysis: Modelling the World in Data,** YOURDON PRESS, Englewood Cliffs, NJ, 1988.

[Str87] STROUSTRUP, B., **The C++ Programming Language,** ADDISON-WESLEY, Reading, Ma, 1989.

[Tel89] TELLO, E., **Object-Oriented Programming for Artificial Intelligence. A Guide to Tools and System Design.** ADDISON-WESLEY, 1989.

[War89] WARD, P.T. , **How To Integrate Object-Orientation with Structured Analysis and Design,** IEEE SOFTWARE, Vol. 6, No. 2, 1989, pp. 74-82.