ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

A Simplified Approach to Control System Specification and Design Using Domain Modelling and Mapping.

G.A. Ludgate
TRIUMF, 4004 Wesbrook Mall, Vancouver, B.C., V6T 2A3, Canada

Abstract

Recent developments in the field of accelerator-domain and computer-domain modelling have led to a better understanding of the "art" of control system specification and design. It now appears possible to "compile" a control system specification to produce the architectural design. The information required by the "compiler" is discussed and one hardware optimization algorithm presented. The desired characteristics of the hardware and software components of a distributed control system architecture are discussed and the shortcomings of some commercial products.

I. INTRODUCTION

In recent years more emphasis has been placed on the gathering and validating of requirements for automated control systems before they are built [1, 2, 3, 4, 5]. Our earlier work reported on the specification of the KAON Factory Central Control System (KF-CCS), using two emerging techniques in the application of object-oriented principles to requirements specification namely: domain driven modelling [2] and dynamic object modelling [3]. A re-examination of the problems encountered using these two contemporary techniques has led to a better understanding of both the use of domains in creating and structuring a system specification and of the design-processes used to transform the system specification into executable code.

It now seems possible to "compile" a control system from its specification form. As with all "compilers" the "target language" (usually a micro-processor machine code) must be *exactly* described before the "compiler" can be created. With contemporary domain modelling approaches we now have the power to construct such a complete description of the active elements that form control systems and to determine an appropriate strategy for "compilation".

II. PROBLEMS WITH EARLIER APPROACHES

Dynamic object modelling [3] advocates, from the outset, the determination of the context of a system-to-be-built and its presentation in the form of a Context Diagram (Fig. 1). A Context Diagram follows from an analysis of a problem, the specification of a solution and the desire to implement the solution as an automated system. This approach leads to an early identification of external devices (Terminator Objects) that are to be interfaced to and controlled by the system. Only information flows between the system and the Terminator Objects are shown on a Context Diagram.The single bubble represents the system-to-be-built; boxes represent Terminator Objects.

The internal structure of the system-to-be-built, termed the Object Communication Diagram, is comprised of both the static and dynamic system-objects in the solution. Figure 2 shows an example internal structure for Fig. 1. Static objects are representation of conceptual entities in the solution (e.g. schedules, lists etc.) while each dynamic object represents the dynamic behavior of its associated Terminator Object, as seen through their mutual interface (the information flows between the dynamic object and the Terminator Object).

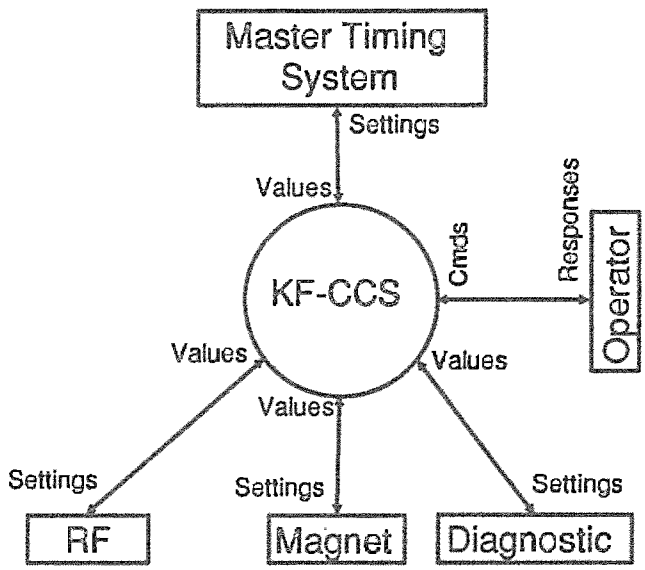

Fig. 1 A simplified Context Diagram of the KF-CCS. The single bubble represents the system-to-be-built; boxes represent Terminator Objects.

During the KAON Factory Study the premature focussing on KF-CCS systems-analysis [3] led to some difficulties with users and reviewers appreciating the role of the (predominantly software based) KF-CCS in the much larger context of controlling KAON Factory beam production, and the affect of interactions between its Terminator Objects. A Super-Context Diagram was therefore created to show the "bigger picture" (Fig. 3). In the KAON Factory, interactions between Terminator Objects will be due, for example, to the Master Timing System that will determine (predominantly in hardware) the exact timing of all beam related events e.g. beam transfers between any of the 5 rings.

Domain driven modelling [2], on the other hand, does not *require* this premature move into systems-analysis. In the early

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

phases of KF-CCS planning a more general form of modelling would have been extremely helpful in understanding the operation of the KAON Factory. With this knowledge in hand, one or

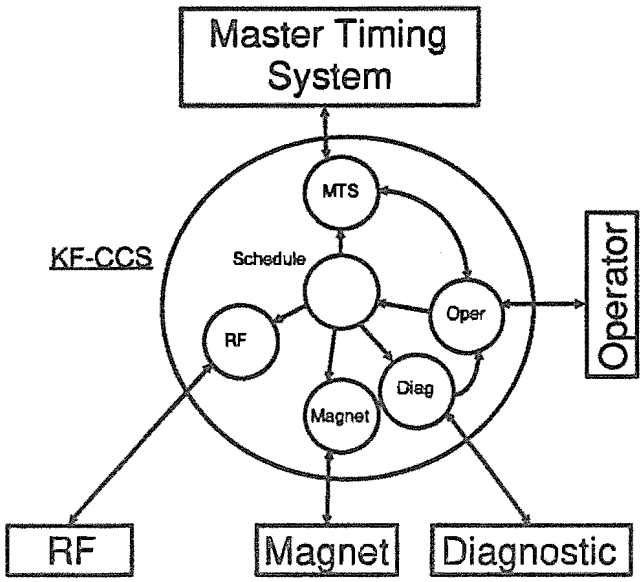

Fig. 2 The Object Communication Diagram for Fig. 1 (inside the bubble) represents the composition of the KF-CCS in terms of system-objects and the information flows between them and the external Terminator Objects.

more systems analyses could have been performed to ascertain the suitability of automating various tasks as part of the KF-CCS.

Models that allow the precise description of *what* is required to be accomplished in a particular field of interest and *when*, but regardless of *how* it is to be carried out, are termed domain models. A complete KAON Factory domain model would show the desired behavior of the KAON Factory regardless of whether control systems had been constructed to achieve the behavior or whether the devices naturally embodied the required behavior. To precisely define the nature of domain models and their use in control system building, a brief introduction to domains follows.

III. DOMAINS

One way humans organize their knowledge of the world is in terms of domains e.g. the domains of banking, physics, art, law, etc. A domain serves as a context within which technical terms and expressions usually have a single meaning; for instance, the expression "*The grounds are fine.*" has completely different meanings in the domains of electronics, gardening, debating and coffee making. The term "domain specific language" highlights this re-use of old words, with new meanings, as an appropriate manner of describing a domain.

Domains can include other domains (e.g. divorce laws are part of all laws), can overlap other domains or be completely independent of each other. The contents of a domain are determined by a set of criteria termed Domain Criteria. It is implicitly understood that the criteria for common-place domains are known to most people. Thus, for example, no-one would consider "*Magnets*" as being part of "*Banking*".

We can capture the essential details of a domain by making models. These models take many different forms, for example:

- written text, for qualitative domains like medicine, or
- written text with embedded mathematical formulae, for quantitative domains like physics.

These models usually describe a domain in terms of five important types of knowledge, namely:

- concepts used by experts working within that domain,
- facts about the individual concepts,
- facts relating two or more concepts,
- interactions that occur, and
- events and conditions that cause interactions within the domain.

This time honoured approach to describing domains, in terms of the "things" that experts believe are part of the domain and the relationships between them, has recently been called "the object-oriented approach" by designers and builders of software systems. The object-oriented approach has been gathering substantial support from vendors and builders alike in recent years due to its uniform manner of modelling both problems and

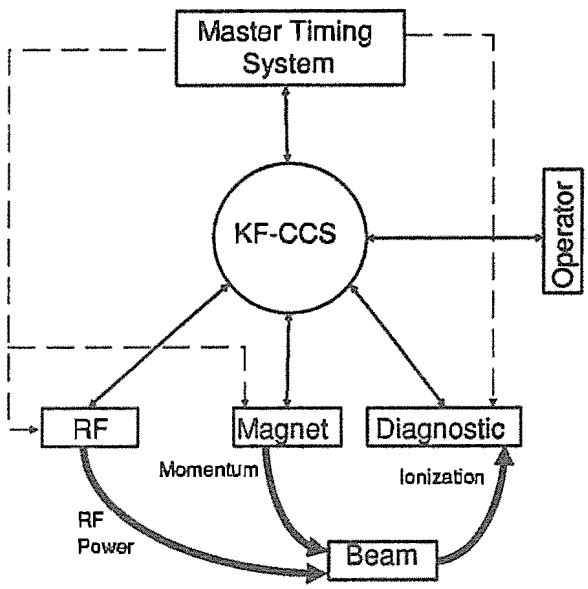

Fig. 3 The Super-Context Diagram of Fig.1 illustrates the "fit" of the KF-CCS into its immediate environment.

solutions, and the availability of languages that support "software objects". The support has been a "grass roots" movement, arising initially from an understanding of the benefits accrued by using object-oriented programming languages on a project; and later expanding to encompass the earlier analysis and design modelling phases of a software project.

IV. DOMAIN BASED MODELLING

One can "view" domain based modelling as being one more step down the historical path of increasingly structuring systems

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

during their construction. Originally, coding was performed in machine code; later in assembler and today in FORTRAN-like languages; structured programming, structured design, objects and applications were more recently created to manage the complexity of large software systems.

A more general understanding of this structuring can be obtained by "viewing" it as successful attempts at constructing new domains with associated domain specific languages. The elements of each new domain were abstractions of useful features from older domains; thus Structured Design highlighted the composition of *programs* from *procedures calling* other *procedures* and *passing data/control couples*. Its approach was language independent in that it abstracted away the details of any procedure's implementation but maintained the usefulness of being able to consider groups of statements as producing definite results from given inputs.

All traditional and most contemporary, commercially available object-oriented development methodologies and languages are presently not domain based in that they do not recognise the existence of domains and their Domain Criteria. In solving a particular problem it is, therefore, possible to incorporate any object what-so-ever into the solution domain. It would strike most of us as being unusual if a "Banking" system specification required "Magnets" but it does not seem unlikely that a "Control System" specification should require "VME".

Both of these examples violate the basic structuring tenet of any domain driven methodology that attempts to prevent the mixing of elements of a given problem (the problem domain) with elements of a particular solution (the solution domain).

Most control systems builders support this domain-driven development heuristic but express it in terms of "layered" designs for control systems - a layer's contents, ideally, being modifiable without affecting adjacent layers [6,7]. In such "good" designs, elements from one layer do not appear in other layers e.g. *network links* inside the *user interface layer*.

Modelling in a scientific domain has often led to a formal mathematical representation of the relationships between the important concepts in the domain but to a completely informal treatment of the 4 other types of knowledge. Physics is a prime example of this modelling approach.

The object-oriented approach to modelling in a domain seeks to redress this imbalance and reduce the emphasis on relationships. To achieve this goal, object-oriented domain analysis embodies a number of model-types that are unfamiliar to physicists and to most builders of automated systems. The models-types that capture the five types of knowledge, cited above, are:

- Extended Entity Relationship Diagrams (EERD); to model the entities and concepts in a domain, their properties and the relationships between them,
- State Transition Diagrams (STD); to model the modes of behavior of each entity and the causes of transitions between the modes,
- Object Structure Diagrams (OSD); to model the processes inherent in each entity and how these effect entity properties and behavior,
- Object Interaction Diagrams (OID); to model the causal relationships between the entities.

Domain models are, therefore, models of an entire field of knowledge in the same way that "PV=nRT" is a physicist's model of an Ideal Gas (a model which could be incorporated into an object-oriented domain model of Ideal Gases as "viewed" by physicists).

Domain models can only be validated by:

- experimenting with physical entities in the domain (e.g. RF cavities and Magnets for KAON), or
- questioning domain-experts about conceptual entities in the domain (e.g. a Beam Schedule, Startup Sequence),

and comparing the results with predictions from the domain models. In like manner the completeness of domain models can only be established by consulting a domain expert; but once the domain models are deemed complete and valid they can serve as a re-usable resource for projects undertaking the automation of activities in the domain. Domain models serve as the *only* criteria against which the "correctness" of any automation project is established.

V. MODELLING IN A DOMAIN

To manage the complexity of modelling in a given domain an observer should represent the domain from a particular viewpoint. Those features that the observer deems *essential* to the viewpoint must be included in the models while all other irrelevant features are omitted. The resulting model is a particular abstract view of the domain. The most important viewpoints of a domain, termed the Canonical Domain Views, are those of the different types of people working in the domain.

The domains relevant to the builders of the KF-CCS were most easily identified from an analysis of all personnel that will be involved in the production of the 30 GeV proton beams [3]. As personnel are frequently assigned multiple jobs, the study focussed on eliciting the roles to be played by those personnel in producing beam. The view of the KAON Factory perceived by any person acting out a particular role defines a description of some relevant (canonical) domain. As several roles often share a common or similar view of the KAON Factory it follows that the number of Canonical Domain Views is limited by the number of roles identified.

An example KAON Factory beam-delivery domain-model (Fig. 4) clearly shows the Master Timing System (MTS), its interaction with each beamline device and with an Operator. Personnel assuming the latter role use the MTS to tune the beam in the synchrotrons after all beamline devices have been turned on. Unlike in Fig. 1, this simple domain model does not distinguish between those functions carried out by hardware or software, and corresponds to the way Beam Physicists describe the operation of the KAON Factory.

A complete model of the domain, formed from a union of the Canonical Domain Views, is termed the Canonical Domain Model.

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

VI. CANONICAL DOMAINS RELEVANT TO CONTROL SYSTEM BUILDERS

In the KAON Factory domain, which is concerned with delivering a 100 microampere proton beam to 30 GeV, the different types of personnel involved [3] have been used to identify several canonical domains, for example:

- the beam-delivery domain; in which KAON Factory Operators and Beam Physicists concern themselves with producing an optimum 30 GeV proton beam and gaining an understanding of the synchrotrons,
- the equipment management domain; in which Equipment Specialists concern themselves with monitoring and controlling devices required to deliver the proton beam,
- the KF-CCS equipment management domain; in which KF-CCS electronics technicians concern themselves with maintaining the equipment related to the KF-CCS operation,
- the KF-CCS implementation domain; in which KF-CCS Managers, Analysts, Designers and Programmers concern themselves with the construction of software according to specifications deduced from the above domains.

Clearly there are relationships between these domains. The last 2 domains are solution domains for problems arising in the first 2 domains, but not vica versa; that is, in trying to implement the required behavior of KAON Factory beam transport devices, when delivering beam, it is easiest to employ ***intelligent*** control system technology to control relatively ***un-intelligent***, passive beam transport devices.

We noted earlier that the requirements for the KF-CCS could have been derived by performing a systems analysis on a relevant KAON Factory domain model, had it been available. Similarly, the requirements for the KF-CCS ***implementation*** could be derived by performing:

- a domain analysis of control systems implementation technologies, followed by
- a system analysis of the domain to highlight the separation of control system functions between available hardware and software.

The latter analysis was performed during the 1989 KAON Factory study for hardware costing purposes, while the former analysis is presently underway.

VII. CONTROL SYSTEMS IMPLEMENTATION DOMAIN

Control systems experts are well aware of their domain of expertise. Figure 5 shows an (incomplete) EERD of a traditional, generic central controls system implementation domain, highlighting the major entities in the domain and some of the relationships between them. A short explanation of the EERD follows with the entities capitalized for ease of reference the first time they are referred to.

All working DESIGNERs and PROGRAMMERs use WORKSTATIONs to create the source classes from the CONTROL SYSTEM SPECIFICATION. The implementation language, represented by COMPILER, is object-oriented, supporting the separate compilation of SOURCE CLASSes that are then linked into an EXECUTABLE IMAGE by the LINKER. The executable images are run within TASKs by a CPU in a PROCESSOR MODULE. They communicate with each using ITCs (inter-task communication links), if on the same processor, and LAN LINKs if on different processors.

Each beam transport and beam diagnostic DEVICE is interfaced to the processors by a single DEVICE I/O MODULE placed in a VME CRATE

Processors have 2 types of storage: CPU MEMORY inside the processor and disks mounted in DISK MODULEs. The former provides volatile storage that is lost whenever power fails, the processor fails, the processor is removed etc., while the latter provides persistent storage that survives such an occurrences.

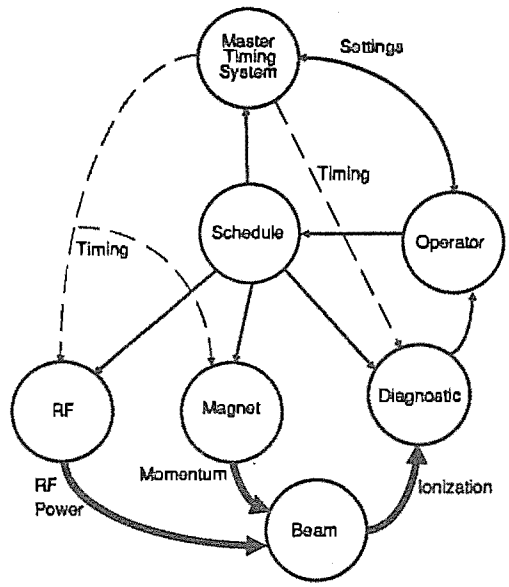

Fig. 4 The Object Interaction Diagram for Fig. 1. This domain model shows the entities in the domain as circles. Heavy, directed lines are energy flows while thin-solid and dashed lines are information flows.

VIII. MAPPING

The CONTROL SYSTEM SPECIFICATION entity (top right of Fig. 5) is a model of a control system to be built. To simplify discussions we will postulate this to be fully described by only one model type, namely an Object Communication Diagram (OCD); Fig. 2 for example. The OCD is, in turn, composed of:

- a set of SYSTEM-OBJECT MODELs,
- a set of SYSTEM TERMINATOR MODELs, and
- a set of all INTER-OBJECT FLOW MODELs describing information flows between System Objects and between System Objects and Terminator Objects.

In addition, each SYSTEM-OBJECT MODEL describes:

- the INTRA-OBJECT FLOWs accepted by the object,
- the PROCESSes required to convert input flows to output flows and

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

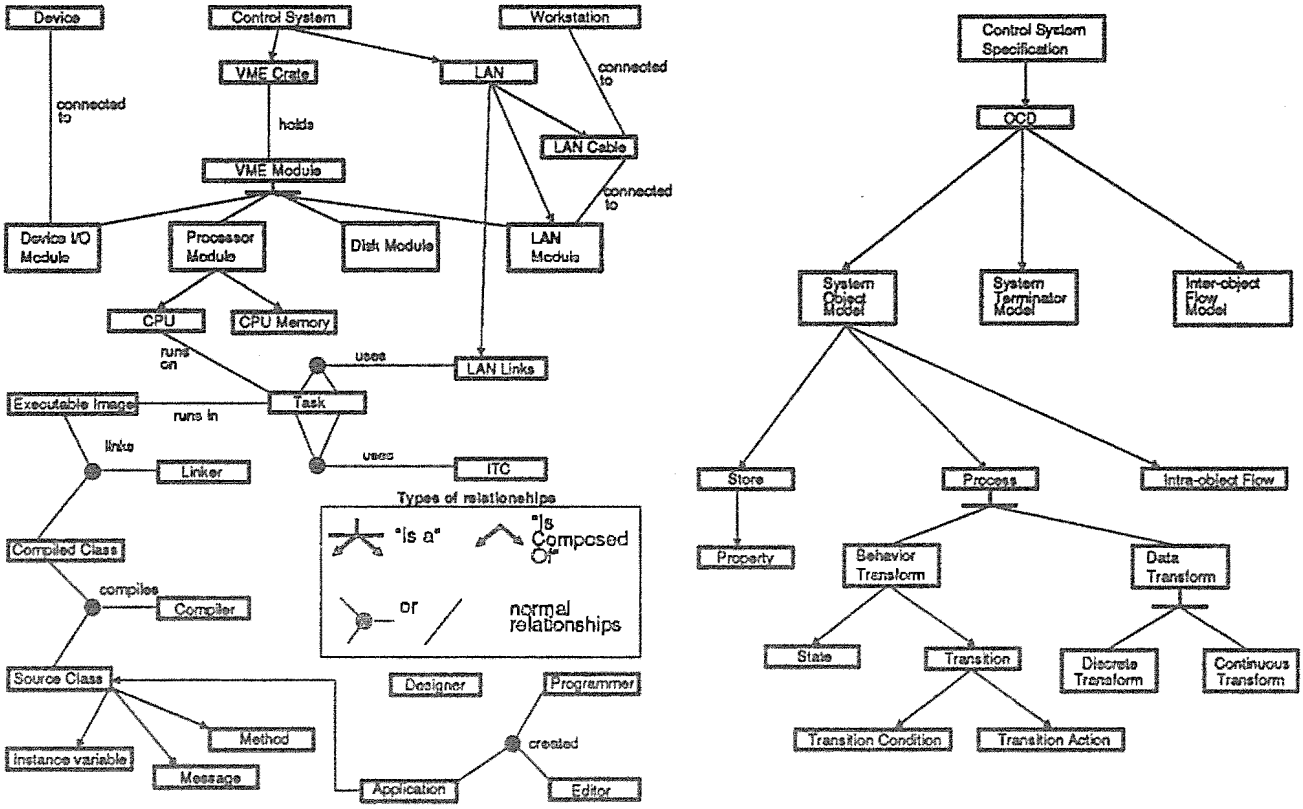

Fig. 5 An (incomplete) Extended Entity Relationship Diagram of a traditional controls system implementation domain. Entities in the domain are represented by boxes. A legend describes the type of relationships between entities.

- the information needed be STOREd to satisfy the PROCESSes.

Each of the above models is a description of a particular type of *need* that entities in the implementation domain must fulfil. That is, we must find entities in the implementation domain that are capable of performing as required by the models. This association of a model with an implementation entity is termed *mapping*. Performing a mapping (i.e. translating) is similar to compiling the description of an algorithm, in a high level language, into an equivalent description in an assembler language. The latter could , in turn, be assembled, linked and executed.

To illustrate the notion of mapping we will consider the various ways of implementing a CONTROL SYSTEM SPECIFICATION using our suite of implementation entities (Fig. 5).

When all system-object PROCESSes are of the discrete variety and we are using a traditional processor, we can map (denoted by the "-->" symbol) these models as follows:

- both types of "flows" --> the MESSAGEs accepted by a SOURCE CLASS,
- "processes" --> the METHODs of a SOURCE CLASS (a named-method is invoked by the class receiving a message with the same name), and
- "stores" --> the INSTANCE VARIABLES of a SOURCE CLASS,
- "system-object model" --> SOURCE CLASS,
- "control system specification" --> APPLICATION, and
- "terminators" --> DEVICE I/O MODULEs.

This mapping implies:

- the control system software is a single application program,
- that it deals with events in the real world sequentially, and
- that all terminator I/O modules must be in the same VME crate.

In a large control system this latter implication is unacceptable. The distributed nature and quantity of terminators will require a large number of terminator I/O modules in several VME crates to interface them to the application software. A local processor will also be required in each crate to access the I/O modules in that crate. The relevant application software must also be running in that processor.

From the OCD of Fig. 2 one can see that the simplest choice of application software to be run in a particular processor must correspond to the system-objects for the terminators interfaced into that crate. Flows between each terminator and the processor would then occur within the single crate.

With this choice of allocating system-objects to processors the original single program now becomes fractured into several programs. Now flows between these distributed system-objects cannot all be mapped to messages; those between objects on different processors must be mapped to LAN LINKS.

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P02

Now consider the changes required when some PROCESSes are CONTINUOUS TRANSFORMs. As these processes must be able to proceed independently they must be mapped to TASKs. In addition the INTRA-OBJECT FLOWs to these processes must be mapped to ITCs (inter-task communication channels). Now a single system-object may be distributed across several tasks, and there is no reason for these tasks to reside on a single processor. If a system-object is fractured across processors then its INTER- and INTRA-OBJECT FLOWS must be mapped to ITCs and LAN LINKs.

IX. PERFORMING AND EVALUATING THE MAP

Given an application program, implemented as a set of traditional communicating programs running on a set of processors, there are many reasons for selecting a particular arrangement of programs to processors. Criteria such as:

- access to critical resources e.g. particular I/O modules,
- communications volume to and from critical resources,
- response times to events,
- meeting deadlines (in hard real-time systems),
- computational "volume" per mode of operation,
- computational "volume" per event,
- processor's capability, capacity and reliability,
- storage capability, capacity and reliability,
- network capability, capacity and reliability,

play important roles in determining the positioning of programs, regardless of operating system and language issues.

The mapping process, described above, leads ultimately to the allocation of parts of SYSTEM-OBJECTs to multiple (sub-application) programs. To simplify the establishment of this mapping, initially, we have studied the case in which:

- system-objects were not fractured smaller than individual PROCESSes, and
- all PROCESSes communicating with a terminator via its I/O module were allocated to processors in the same crate.

The number of application program fragments to distribute is then large but finite. We are presently investigating an algorithm that would produce a "best" arrangement of programs among processors by minimizing a "quality factor". The "quality factor" depends on the values of all of the properties listed above. These values will all be available from a control system implementation-domain model presently being completed for the KAON Factory.

The simple mapping model presented here has difficulties accounting for certain properties required by some systems - for instance "fault tolerance". In a fault-tolerant system one would require copies of a program to execute on different processors - a feature that would increase the amount communications over a solution in which all copies were allocated to the same processor. The role of a processors "reliability" property in the "quality factor" must be, therefore, to "repel" copies of like software.

X. CONCLUSIONS

The possibility of generating a mapping directly from a control system specification now seems feasible. It relies upon:

- detailed domain and system models of the control system implementation architecture (e.g. Fig. 5) and either
- a detailed mapping scheme that captures the experience of control system designers, or
- a calculable figure of merit associated with each mapping that could be minimized, say, to produce the "best" design.

Commercial code generator products have been available for several years that are capable of creating executable systems directly from structured designs. They are feasible because the commercial computing domain is very mature and code generators tend to work with mainstream transaction management products for which all interface software is provided.

It is timely to pursue the goal of developing control systems at a higher "level" of abstraction. There are many benefits to be gained by leaving the "technical details" to a new form of "compiler" in much the same way as assembler programs were replaced by high level language programs. The final products will be more uniform in design and more adaptable to technology changes by "re-compilation". Future control system development work at TRIUMF in support of the KAON Factory will be directed toward developing such a "control system compiler".

XI. REFERENCES

[1] D.A. Dohan, G.A. Ludgate, E.A. Osberg, S. Koscielniak and C. Inwood, "Definition study of the TRIUMF KAON Factory control system project", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 6-11.

[2] C. Inwood, G.A. Ludgate, D.A. Dohan, E.A. Osberg and S. Koscielniak, "Domain-driven specification techniques simplify the analysis of requirements for the KAON Factory central control system", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 390-393.

[3] E.A. Osberg, G.A. Ludgate, S. Koscielniak, D.A. Dohan and C. Inwood, "Dynamic object modelling as applied to the KAON control system", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 394-401.

[4] J-L. Theron, "Design and checking of a large Ada real-time system", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 373-376.

[5] G. Morpurgo, "SASD and the CERN Run-time coordinator", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 385-389.

[6] B. Kuiper, "Accelerator controls at CERN: some converging trends", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 308-315.

[7] P. Clout and A Daneels, "Report of the Los Alamos accelerator automation application toolkit workshop", Proceedings ICALEPCS '89, Nucl. Instr. and Meth. A293 (1990) 321-324.