IPTES – Incremental Prototyping Technology for Embedded real-time Systems

Petri Pulli
Technical Research Centre of Finland (VTT)
Computer Technology Laboratory
P.O. Box 201, SF-90571 Oulu, Finland
E-mail: ppm@vtttko.vtt.ii

René Elmstrøm (Contact person)
The Institute of Applied Computer Science (IFAD)
Munkebjergvænget 17, DK-5230 Odense M, Denmark
Tel: +45 65 93 43 03
Fax: +45 65 93 29 99
E-mail: rene@ifad.dk

Gonzalo León and J.A. de la Puente
Dpto. de Ingenieria de Sistemas Telematicos
Universidad Politecnica de Madrid
E-mail: gleon@dit.upm.es, jpuente@dit.upm.es

August 24, 1994

Abstract

The constantly increasing complexity and risk associated with the industrial development of embedded computer systems has been approached in different ways in resent years. One of the most promising approaches to managing risks in software development projects is the Boehm's spiral principle.

In ESPRIT project no. EP5570 called IPTES (IPTES is an acronym for "Incremental Prototyping Technology for Embedded real-time Systems") a methodology and a supporting environment to support the Boehm's spiral principles are being developed. The prototyping environment will enable the specification and verification of executable system models so that different parts of the system may represent different modeling levels, and yet they can be executed as a total system. Also problems in connection with distributed software development are addressed in the IPTES environment. Typically it is difficult to define the interface protocols between the processors (nodes) of a multi-processor real-time system. In the IPTES project the concept of distributed prototyping is proposed as a solution. Each development team working on their processor (node) can test their own physical model against the logical model of the rest of the system.

The IPTES environment provides a set of modules to help in the process of creating, analyzing and testing distributed heterogeneous prototypes. Internally, the environment is based on a representation of the system in terms of high level time Petri nets with shared places.

The interchange of information between the nodes is performed by a real time object communication (RTOC) subsystem that ensures consistency between shared places.

1 Introduction

Three major trends can be seen in the development of software for embedded systems in the nineties:

1. The complexity of systems is constantly increasing
2. The number and the loss impact of risks associated with software development are on the increase
3. Just-In-Time delivery policies and flexible manufacturing will require new degrees of freedom from the traditional software development process

These trends leads to the need for stronger and more systematic system description notations. In general, the software part of typical embedded systems is getting more dominant. Therefore the management of risks like: schedule overruns, man-power or budget overruns, functional or quality pitfalls or in the worst case canceled projects is getting still more important in the industrial development of embedded systems.

In the last few years more and more attention has been paid to alternative software develop-Mont lifecycle models that could better take into account complexity and risk management. The risk-driven nature of the Boehm's spiral software process model makes it particularly applicable to the construction of complex embedded systems.

The spiral approach leads us to postpone detailed elaboration of low-risk elements and to avoid going too deep in their design until the high-risk elements of the design are stabilised. The spiral approach incorporates prototyping as a risk reduction option at any stage of development. It also accommodates reworking, or returns to earlier stages, as new options are identified or as new risk issues need resolution. The identification and evaluation of options encourages the reuse of existing, proven components.

In the IPTES project methodological guidelines, theoretical results and a supporting prototyping environment are developed to support Boehm's spiral principles.

We use the term *incremental prototypes* for the system models supported in the IPTES environment. Incremental prototypes are heterogenious and distributed and support an incremental change of any subpart to the next (or the previous) maturity level down to a level of source code. Source code is integrated into the system models through run-time adaptation technology.

This principle will make it possible to specify executable system models on different abstraction levels and to execute and analyse system specifications where the maturity of component specifications varies from high-level logical models to final software code components.

In the next section we will present some related work, including other tools supporting heterogenious prototyping. In section 3 we will present the general idea of prototyping as a means of supporting the Boehm's spiral principles. In section 4 we will introduce the general approach of the IPTES project including the different types of system models supported in the IPTES environment. In section 5 we give more details on the proposed architecture of the IPTES environment and finally we give some concluding remarks.

2 Related work

Gabriel presents the requirements for a future prototyping environment in [Gabriel89]. Gabriel foresees the need for heterogeneous prototyping by requiring that elements from behavioural and structural prototypes can be combined and during the development process this mix may change as the requirements to the prototype change.

Luqi has presented a prototyping environment for large software system design based on reusable Ada software components [Luqi86], [Luqi&88], [Luqi89]. The computational model is based on data flow under semantically unified control and timing constraints. Luqi has presented the importance of the computational model for a prototyping tool, language and method in

[Luqi86]. A limitation of the Luqi's system is that it does not support multiple abstraction levels. The abstraction level supported is roughly equivalent to the software environment model of SA/RT[1].

Harel et al. have produced a commercial, graphical executable specification tool, Statemate [Harel&90], that has prototyping features. It is possible to automatically generate prototype code from the activity-chart and statechart [Harel87] specifications. Currently translations into Ada and C code are supported. A limitation of the Statemate tool is that it does not support multiple abstraction levels. Currently, an abstraction level equivalent to the logical model of SA/RT is supported[2]. However, it is possible to combine the prototype code generated out of Statemate models with user-written programs in Ada or C [Harel&90], [Coleman&90].

[Blumofe&88] and [Cadre90] describe a commercial, graphical executable specification tool, Teamwork/SIM. This tool supports a limited form of execution, capable of expressing only control and timing issues. Computations and data have been omitted. However, Teamwork/SIM supports several abstraction levels equivalent to the LM, PEM and SEM from SA/RT.

There are a number of tools for SA/RT logical model execution [Webb&86], [Reilly&87], [Coomber&90], [Athena89].

Some of the surveyed tools can be used for execution of heterogenious models and in that sense they make incremental prototyping possible. However, none of the tools supports incremental prototyping[3].

3 Prototyping within the Framework of the Spiral Model

Over the last few years more and more attention has been paid to alternative software development models that could both overcome deficiencies [Agresti86] of the traditional waterfall model [Boehm81], and accommodate activities such as prototyping, reuse, and automatic coding as part of the process. The spiral model proposed by Boehm [Boehm88] (Figure 1) is a major step into this direction. The risk-driven nature of the spiral model makes it particularly applicable to complex embedded systems.

3.1 Risk Management

The spiral model guides a developer to postpone detailed elaboration of low-risk software elements and to avoid going too deep in their design until the high-risk elements of the design are stabilised. Risk management requires appropriate attention to early risk resolution techniques such as early prototyping and simulation. The spiral model may incorporate prototyping as a risk reduction option at any stage of development and explicitly calls for suitable risk assessment and risk control activities throughout major portions of the development process.

Risk management involves the following steps [Boehm91]:

- Risk assessment techniques
  - Risk identification produces lists of the project specific risk items likely to compromise a project's success.
  - Risk analysis quantifies the loss probability and loss magnitude for each identified risk item, and it assesses compound risks in risk item interactions.
  - Risk priorisation produces a ranked list of risk items according to their severity.
- Risk control techniques

[1] One can argue that there are in fact two abstraction levels: the Software Environment Model level and the code level. This is because Luqi's system does not have a mini-spec language, instead Ada is used.

[2] [Harel&90] mentions plans to make the prototype code generation more adjustable. This can be interpreted as an indication of interest to support more physical abstraction levels.

[3] To better understand the difference between "makes possible" and "supports", consider the case of object-orientation. Any programming language can be used object orientedly, however, the benefits of object orientation were first realised with proper environments like Smalltalk and C++

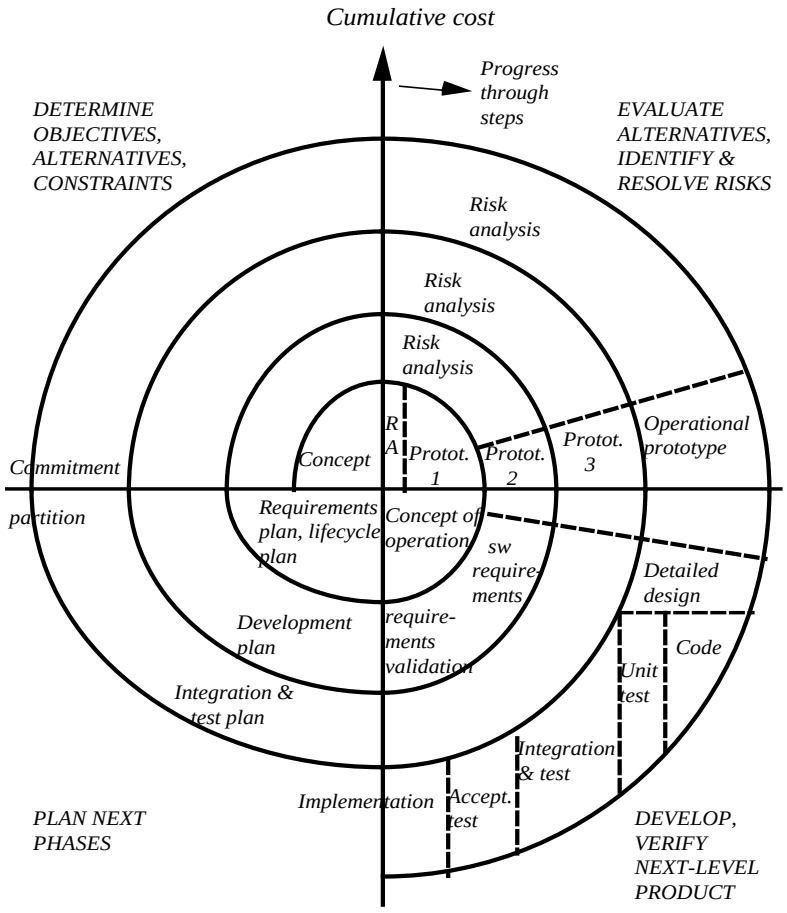

Figure 1: Boehm's spiral model for the software development process. The radial dimension represents the cumulative cost, and the angular dimension represents the progress made in completing each cycle of the spiral.

- Risk management planning helps prepare you to address each risk item. It also includes the coordination of the individual risk item plans with each other and with the overall project plan.
- Risk resolution produces a situation in which the risk items are eliminated or otherwise resolved.
- Risk monitoring involves tracking of the project's progress towards resolving its risk items and taking corrective action where appropriate.

3.2 Concurrent Threads of Activities

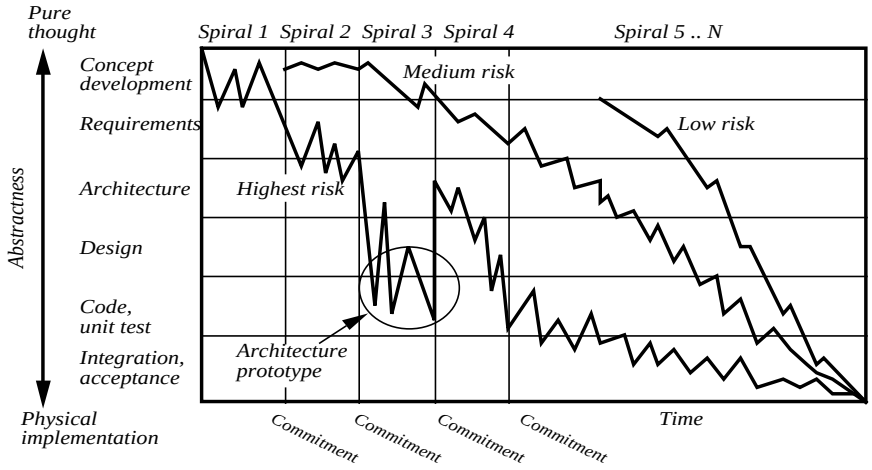

Figure 2: Level of abstraction vs. time under the spiral model.

The spiral model allows concurrent threads of development activities that may traverse the traditional progression of software product phases in a loosely synchronised manner. The concurrent threads may be organised around levels of risk [Boehm88]. Figure 2 gives an example of concurrent development threads [TRW89]. In Figure 2, the horizontal axis represents elapsed time. The vertical axis represents the level of abstraction at which a representation or an understanding of the system is being developed. The development process is depicted by three traces through the two dimensional space. Each trace corresponds to a thread of engineering activities. In general, moving downwards represents progress. A highly jagged trace represents an activity thread in which much iteration (prototyping) takes place. This is seen in Figure 2 where an architectural prototype is developed for the high-risk thread, during the third spiral. This prototype is later analysed and thrown away. Based on the lessons learned from the architectural prototype, the high-risk thread enters full-scale design and implementation during Spiral 4. Concurrent with the third and fourth spirals the medium-risk elements are being specified. During the fifth and later spirals the high-, medium- and low-risk threads progress concurrently, leading to incremental integration, installation, and use.

3.3 Abstraction Levels

The spiral model is a generic model, so it does not explicitly define the milestones to be produced for each cycle. The spiral has to be customised on a company or project basis. [Royce90] presents a derivative of the spiral model which explicitly defines milestones according to the US military standard [DOD-STD-2167A].

Figure 3 presents an example of typical abstraction levels of a well-known embedded system development method, Ward & Mellor's Structured Analysis for Real-Time Systems (SA/RT) method [Ward&85]. These abstraction levels are described using the same modeling languages:

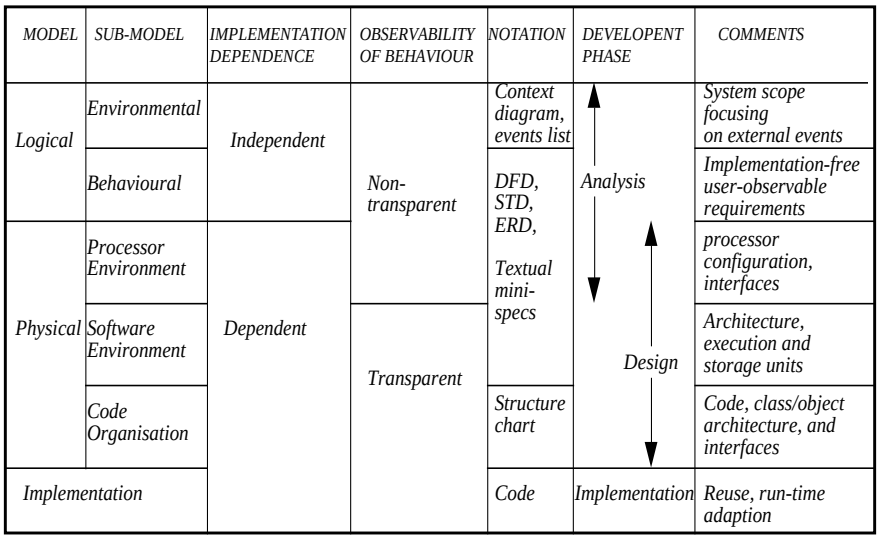

Figure 3: The different models for real-time system development proposed by Ward & Mellor.

data flow diagrams (DFD), state transition diagrams (STD), and entity relationship diagrams (ERD).

The Logical Model (LM) consists of an environmental model and a behavioural model. In the environmental model the system's environment and the events from this environment are described. In the behavioural model the system's externally observable behaviour is described. The physical model consists of several sub-models. The Processor Environment Model (PEM) describes how system activities and data are allocated to different processors which are truly concurrent. The interfaces between processors are also described. The Software Environment Model (SEM) is a description of the software architecture inside one processor. The SEM model describes how concurrency will be solved using a sequential processor. The Code Organisation Model (COM) describes the modularisation scheme to implement the software. It will identify a hierarchy of modules and data structures. For object oriented designs it will identify a hierarchy of classes and objects. The Implementation Model (IM) describes the product implementation.

4 IPTES approach

4.1 General goal

The IPTES approach to support the Boehm's spiral principles is to offer a prototyping environment that will enable a project team to develop software for embedded systems incrementally. Executable system models whose different parts represents different abstraction levels can, in this environment, be developed and executed as one system. In this way a high-level logical model of a system may first be developed and the refinement of this model can be guided by a risk-analysis of the different parts of the model.

The executable system models, or prototypes, supported by the IPTES environment has three main properties: they are

- heterogeneous
- distributed, and
- they support incremental change of any subpart to the next (or the previous) maturity level.

We use the term *incremental prototype* for prototypes with these properties.

4.2 Heterogeneous Prototype

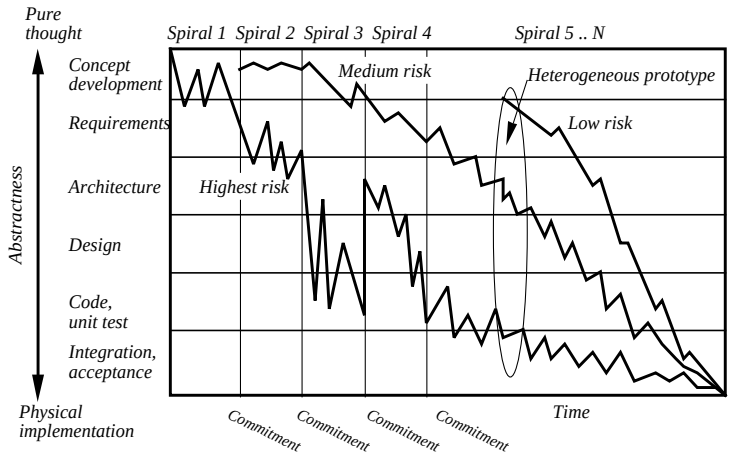

Figure 4: A heterogeneous prototype can be viewed as a vertical snapshot of concurrent development threads of Boehm's spiral model.

A heterogeneous prototype is an executable system model whose different parts may be specified at different abstraction (modeling) levels, and yet they can be executed together as total system. Over the lifetime of the prototype the mix of abstraction levels may change [Gabriel89]. Figures 4 and 5 [Mortensen90] illustrate the concept of the heterogeneous prototype.

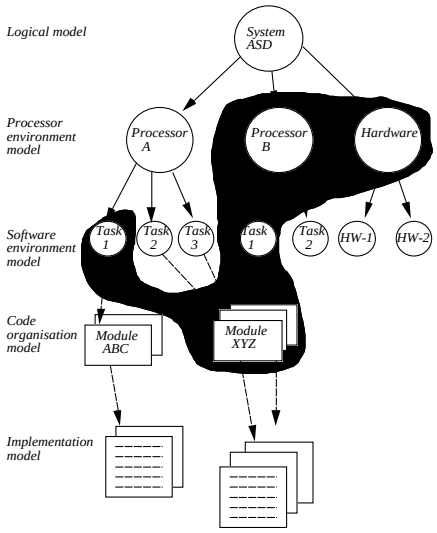

Figure 5: An example of a heterogeneous prototype. The marked area presents a possible coverage of the executable model consisting of partial models belonging to different abstraction levels.

4.3 Distributed Prototype

Embedded systems are often highly concurrent in nature. For very complex embedded systems, the implementation of the concurrency may require distribution of the system activities to several hardware processors that are loosely coupled. System development projects for distributed embedded processor systems are very difficult in terms of controlling both the project and the co-operation of different project teams working on individual nodes of the system. The reason

is that it is very difficult to define the interface protocols between the processors (nodes), and if they are somehow successfully defined they tend to keep changing.

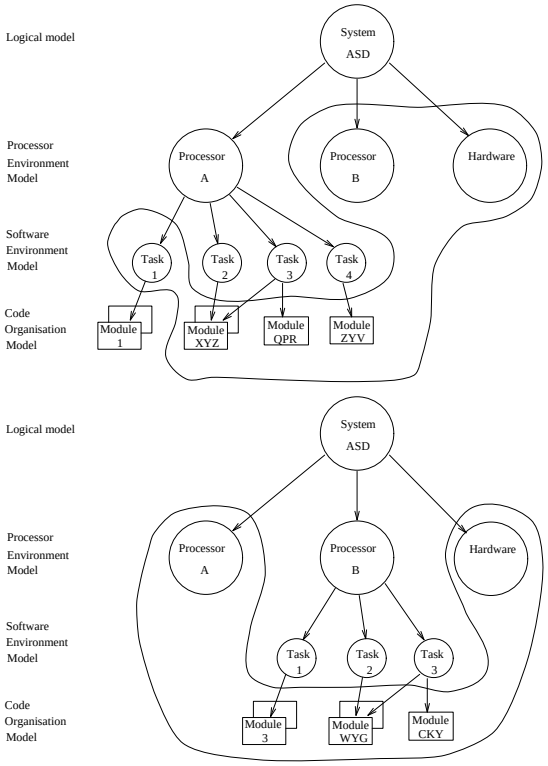

Figure 6: Distributed prototyping. Two different project teams may work simultaneously on separate workstations prototyping different parts of the system against the logical model of the rest of the system located on other workstations.

In the IPTES project the concept of distributed prototyping is proposed as a solution: Each development team working on their processor (node) can test their own physical model against the logical model of the rest of the system, Figure 6. This enables the development of large distributed, even multi-supplier embedded systems in a harmonious way.

4.4 Incremental Prototype

We expand the concepts of heterogeneous and distributed prototyping to the concept of incremental prototyping. Incremental prototyping covers heterogeneous prototyping, distributed prototyping and the possibility to incrementally change any subpart of the model to the next (or the previous) maturity level. In this way incrementally evolving prototypes may be developed and validated, by means of execution, and progress can be made through further refinement of subparts or rejection of the prototype and return to the previous maturity level of the last refined subpart.

4.5 Specification languages

The use of incremental prototyping requires use of executable specification languages. In the IPTES project support for SA/RT graphical specification and design descriptions augmented with VDM descriptions for mini-specifications will be provided, but the IPTES prototyping environment will be open for extension with other executable specification languages by providing an underlying "time Petri net machine" as a basis for the execution and analysis of formal specifications. Run-time adaptation techniques will be used to support an incremental

development of the prototypes into object code and execution of specifications where parts are defined as SA/RT-VDM specifications and parts are refined to a level of object code.

Further aspects of the proposed architecture of the IPTES environment will be presented in the next section.

5 IPTES architecture

5.1 Introduction

To support the IPTES approach, different tools will be designed and interconnected under a common prototyping environment. This section presents the architecture of the environment to be designed during the IPTES project.

In the technical literature there are two different but complementary approaches: the definition of a broad and general scope environment (on a heterogeneous network) to be used as a backbone for many distributed applications, and the design of a highly integrated environment closely adapted to a specific application.

The architecture we have chosen is closely related with the second approach, but on a heterogeneous platform. Nevertheless, that requirement should be compatible with the use of standard tools to ensure an easy migration to different platforms. Industrial standards for the user interface and the integration of tools will be used in those parts where performance requirements are not essential.

5.2 Requirements

The proposed environment should support the design of real-time systems under an incremental prototyping technique. The designer must receive help from the initial phases (requirements capture) to an implementation in a procedural language. The most important requirements concerning the architecture of the environment can be described as follows:

- **High level graphical design language support**

  At least an SA/RT [Lintulampi90b] machine augmented with VDM [Jones90] for mini-specifications, and a notation to represent real-time constraints should be available. This machine provides the high level language for the designer. The designer working at this level is not aware of the internal representation chosen in the inner layers. We will refer to it as the *specification level*.

- **Analysis in terms of high-level time Petri nets**

  The basic kernel to which the different parts of the system should refer is a high-level time Petri net machine. The kernel will be built around an abstract net model supporting the representation of typed data objects and time in a general way. This level will be named the *execution kernel layer*.

  Animation, analysis and reasoning on the system will be made on the kernel description by specific tools. However, the results shall be mapped back to the original specification language used by the designer.

- **Support for distribution**

  The concept of *distribution* in the IPTES environment has two different perspectives: distribution of the prototype itself and the distribution of the design environment.

  The first concept is related to the idea that different parts of the system can be located on different sites (workstations, file servers or specific target systems). The degree of maturity of the different parts of the system is not uniform and the same module (from the user point of view) can approach to its final form (procedural code) progressively with time although the rest of the system keeps its less mature level.

To cope with this problem, virtual processors and physical processors will be considered. In some cases, the location of the parts is transparent to the user; in other cases, above all in relation with the target systems, transparency is not appropriate and the user may decide the location.

The second concept is associated with the possibility that the user can access different tools running on different sites. The distribution support layer should control the activation of the different tools and provide communication mechanisms (data and control) between them.

- **Assumptions about real-time**

  The purpose of IPTES is the development of real-time embedded systems. Real-time systems are defined as computer systems whose semantics depend on time. A real-time system must react to events occurring at unpredictable times within specified intervals. Too early or too late a response may result in unacceptable behaviour of the system.

  The conceptual model of time which will be used to support the real-time functionality is Newtonian time, i.e. physical time in the usual sense, referring to some ideal universal clock with infinite precision. This ideal model can be represented by finite precision clocks, but we shall not take into account any concept of "distributing" time. Data and events which are time-stamped will always refer to universal time.

  Animation of real-time system specifications often requires that time elapses at a different speed than in real life. This is due to the difference in dynamic behaviour between the system and the designer, and to the different execution speeds of the specification and the target system that will be built from it. To solve this kind of problem, a notion of simulated time will be needed in the IPTES environment.

- **Run time adaptation**

  One of the most important objectives in the IPTES project is to facilitate the simultaneous consideration of procedural code and specification code. This requirement will force us to consider interfaces to a global communication interface in terms of objects and operations without reference to their status (as procedural or specification code).

  Accordingly with the system distribution concept mentioned above, specific modules for integrating preexistent code should only be considered in some locations (i.e. where target systems will be present).

- **Common user interface**

  Although there are different tools and the user can act on them during the system prototyping development, it is appropriate to provide a coherent user interface regardless of the characteristics of the different tools.

  Visual integration will be provided by appropriate metatools. The user should specify the location of tools, their relation and the shape in which they are shown to the user. Standard tool-kits can be used.

5.3 IPTES functional view

5.3.1 IPTES design principles

Before considering the proposed modules, general requirements for the environment design and implementation will be outlined. They are related with the general requirements expressed above. These are:

- **Layered architecture**.

  We propose to adopt a layered architecture to simplify the design and to take advantage of preexistent tools. This structure will allow the design in parallel of different tools and the possibility to create virtual interfaces to test a specific module.

The use of the concept of **layer** produces a hierarchical view of the environment where the bottom layer is associated with the hardware and software infrastructure and the top layer is related to the user activities. The services offered in one layer are defined as a consequence of the needs identified by the upper layer (for example, at the specification level). In some cases, those services are mapped down to the inner layers although a one-to-one mapping may not be kept.

- **Open to functional extension**

  As a general design principle we will try to design the tools in a modular fashion where new functionalities could easily be added. This requirement has only effect during the design of the prototypes of the environment.

- **Different levels of integration**

  The ability to include new tools (as a type of functional extension without reference to the previous ones) or to extend preexistent tools, produces a different framework of integration.

  Two levels of integration will be considered: external tools and integrated tools. The first type groups those tools that are not capable of dealing with the objects and protocol supported in the environment. The second group are IPTES tools that will use the high level communication infrastructure provided by the environment.

Figure 7 represents schematically the identified layers. In the following sections a preliminary description of the modules is presented.

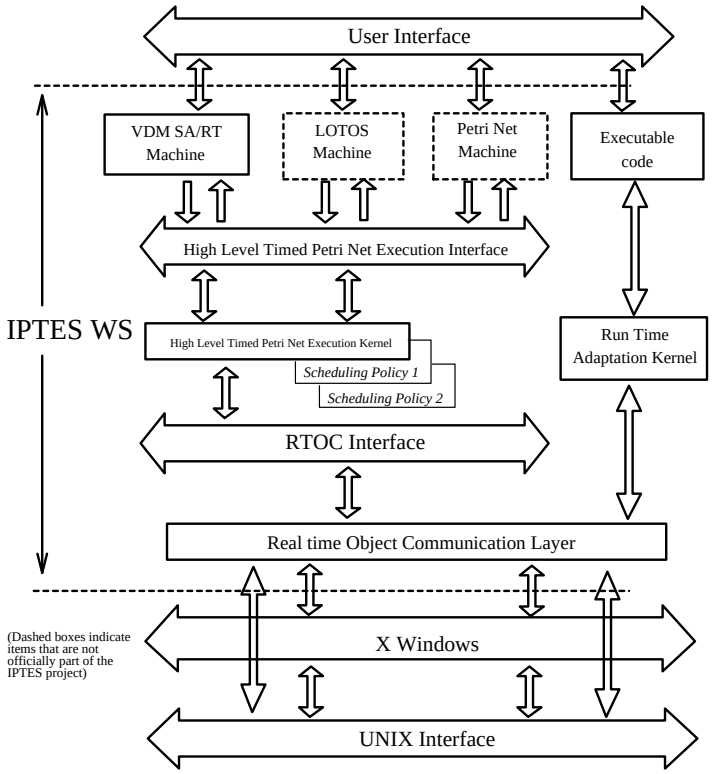

Figure 7: IPTES layered architecture

5.3.2 User interface

This layer provides a common interface from the user standpoint and it provides a visual integration of the different tools. The minimum set of functions is as follows:

- Activation of tools regardless of their location (physical or virtual processor). This function is associated to the transparency of the distribution of the design environment.
- Customization of the distribution of the environment through migration of data or code. This function will allow the generation of the environment on different hardware/software platforms.
- Recording and recovering facilities from libraries (real-time specification components, data and code modules). Nevertheless, the actual tools to store modules could be taken from external tools.
- Integration of new tools by specifying their relation with other preexistent tools.

The user interface will be based on a multi-windowing capability, using a standard software package.

5.3.3 System specification level

The objective is to generate a system prototype in a graphical high level real-time specification language in accordance with the prototyping approach. The minimum set of functions is as follows:

- Graphical specification language support. Several tools could be necessary if different graphical notations are used.
- Conversion to kernel representation (high level time Petri nets). This includes conversion of DFD diagrams and data and time representation to high level time Petri nets (HLTPN).
- Support for model manipulation: Generation of models with different levels of detail (hierarchical modeling with consistency checking between levels), specification of real-time constraints, resource allocation and partitioning of the model.
- Animation of models by means of actions interpreted by the kernel.

The basic high level specification machine will be based on the SA/RT specification language augmented with VDM for descriptions of data structures and data transformations. Initially, the proposed functionalities will be derived from the ESPEX tools [Lintulampi90a] although new versions with enhanced functionality will be designed during the project.

5.3.4 High level time Petri net kernel

High level Petri nets with time will be used as a kernel notation for the IPTES environment. This level has no direct visibility to the user but rather provides an internal framework to be used by all the tools in the IPTES environment. Inverse mappings will be provided in order to represent the results of the HLTPN kernel operations in terms of user notations. Initially, the kernel functionality will be derived from ER net tools [Ghezzi&89]. The kernel will provide functionality for:

- Creating, deleting and modifying nets and subnets.
- Executing net models.
- Partitioning net models into different virtual or physical processors[4].
- Executing distributed net models.
- Execution according to the chosen of scheduling policy.

[4]The partition of the model is the responsibility of the user at the SA/RT level. This partitioning is reflected in the HLTPN kernels which generates orders to move objects at the RTOC levels. Access to partition primitives in the HLTPNK would be interesting but would require knowledge of the Petri net mechanisms.

5.3.5 Run Time Adaptation Kernel (RTAK)

As mentioned above, in some locations a target system will run procedural code. The interaction between this code and the specification level (SA/RT-VDM machines) in other locations will be provided by the run time adaptation module. This layer functions (w.r.t. communication with other layers) exactly as the HLTPN kernel does in other locations. This is the objective of the RTAK.

In short, the functionality is similar to that provided by a dynamic symbolic debugger but controlled in this case for order-objects received from other parts of the system. The minimum set of functions is as follows:

- Managing conversion procedures between simulated and real time
- Packing and unpacking objects sent or received
- Managing pending actions according to a scheduling or queueing policy.
- Controlling the progress of the target system in accordance with the events received by other parts.

5.3.6 Real-Time Object Communication Subsystem (RTOC)

The objective of RTOC is to provide a mechanism to manage and move objects from one abstract (virtual or physical) machine to another in a transparent way. The RTOC layer will provide a set of services to support partitioning of HLTPN models as described above on a distributed system. The minimum set of functions is:

- To ensure the integrity of objects shared by different parts of the system.
- To keep generic objects to be later instantiated
- To create, delete, migrate or modify local and remote objects.
- To implement distributed execution mechanisms for HLTPN. This includes:
  - To transport data objects between subnets.
  - To maintain consistency with respect to simulated time.

Therefore, this layer has two different functions, the cooperation with the HLTPN layer or the RTAK by receiving requests for managing objects (it will be named IPTES object management layer) and the procedures to transport specific objects as messages in a reliable way between IPTES tool-objects (it will be named RTOC protocol layer).

In the previous sections we have mentioned the need to manage the concept of simulated time and real time (on the target system) and the problems derived from it. In this section we will analyze the consequences about the time management for defining the procedure on the real-time object communication subsystem.

To understand the problems involved, we can imagine a situation where two HLTPN subnets on two different locations are running. Both subnets are related (we can assume that the relation is through common places) and when an event happens in one subnet it will be received in the other net. It means that in both subnets there exist lists of pending actions with deadlines for executing.

When one of them needs to consume an action, some type of permission is necessary (as a message to the other subnet) in order to trigger the actions in accordance with a common time scale. As a result of the action a resynchronization of the simulated clocks should be done.

5.4 Prototyping technique

Two prototypes will be used before generating the final IPTES environment.

The first prototype is focused upon the understanding of the problems to be solved to communicate objects residing on the same workstation. Complex functions to manage objects will not be included because they are not necessary in order to understand the RTOC design problems. It will serve to evaluate the design approach and the problems found in its preliminary definition.

The second prototype will extend the functionality by including object management and generation of user interfaces in a distributed framework.

Figure 8 summarizes the evolution from the first prototype to the second one.

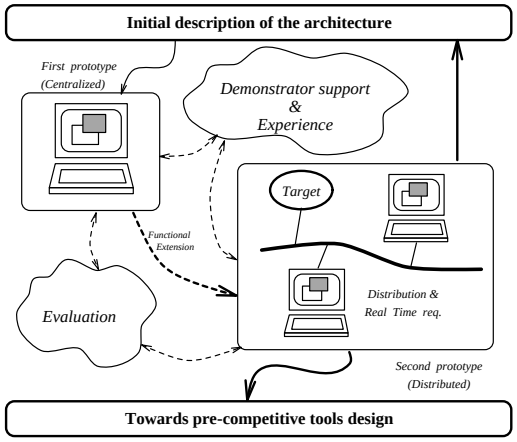

Figure 8: Evolution of the IPTES prototypes.

6 Concluding Remarks

We found it impossible to built an environment for the support of incremental prototyping on top of any existing environment tools because of inadequacy of the internal computational models of the tools. Incremental prototyping requires a computational model able to support at the same time

- all meaningful executable abstraction levels of a software product developed according to a disciplined methodology
- passing data and control between models of different levels of abstraction during execution
- concurrency
- distributed execution
- time

To develop this model of computation and to implement it is a major scientific challenge for the project.

7 Acknowlegdements

The IPTES consortium is formed by IFAD (Denmark), VTT (Finland), MARI (United Kingdom), CEA/LETI (France), ENEA (Italy), Synergie (France), Universidad Politécnica de Madrid (Spain), Telefónica I+D (Spain), and Politecnico di Milano (Italy).

8 References

[Agresti86] W. Agresti (editor). *New Paradigms for Software Development.* IEEE Computer Society Press, 1986. 295 pages.

[Athena89] Athena Systems Inc. Foresight: Modeling and Simulation Toolset for Real-Time System Development, User's Manual. March 1989.

[Blumofe&88] Blumofe, R., Hecht, A. Executing Real-Time Structured Analysis specifications. *ACM Sigsoft Software Engineering Notes*, 32–40, 13, 3 1989.

[Boehm81] B. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981. 767 pages.

[Boehm88] B. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Software*, 21(5):61–72, 1991.

[Boehm91] B. Boehm. Software Risk Management: Principles and Practices. *IEEE Software*, 32–41, January 1991.

[Cadre90] Cadre Technologies Inc. Teamwork/SIM. User's Guide. Release 4.0. December 1990. Part Number D048XX4A.

[Coleman&90] Coleman, G.L., Ellison, C.P., Gardner, G.G., Sandini, D.L., Brackett, J.W. Experience in modelling a concurrent software system using Statemate. In , editor, *Proceedings of the Compeuro'90 Conference. Tel Aviv, Isreal*, pages 104–108, Washington D.C.IEEE Computer Society Press, May 1990.

[Coomber&90] Coomber, C., Childs, R. A graphical tool for prototyping of real-time systems. *ACM Sigsoft Software Engineering Notes*, 70–82, 15, 2 1990.

[DOD-STD-2167A] Anon. *Military Standard DOD-STD-2167A - Defence System Software Development.* Department of Defence, Washington D.C. 20301, February 29, 1988. 51 pages. Obtained From Global Engineering Documents 2805 McGaw Ave., Irvine, CA 92714.

[Gabriel89] Gabriel. Draft Report on Requirements for a Common Prototyping System. *ACM Sigplan Notices*, ():24(3):93–165, 1989.

[Ghezzi&89] Carlo Ghezzi, Dino Mandrioli, Sandro Morasca and Mauro Pezzé. A General Way to put Time in Petri Nets. In *Fifth International Workshop on Software Specification and Design*, pages 60–67, ACM Sigsoft, 1989.

[Harel87] Harel, D. Statemate: A visual formalism for complex systems. *Science of Computer Programming*, 231 – 274, 8, 3 1987.

[Harel&90] Harel, D., Lachover, H., Naamad, A., Pnuelli, M., Politi, M., Sherman, R., Shtul-Trauring, A., Trakhtenbrot, M. Statemate: a working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 403 – 414, 16, 4 1990.

[Jones90] Cliff B. Jones. *Systematic Software Development Using VDM (second edition).* Prentice Hall, Englewood Cliffs, New Jersey, 1990. 333 pages.

[Lintulampi90a] Raino Lintulampi. ESPEX – Executable Specification Based Validation and Incremental Prototyping Environment. Presented at TCC2, May 1990. 18 pages. slides.

[Lintulampi90b] Raino Lintulampi. *A Specification of the Simulation of SA-RT Models.* Technical Report, VTT, November 1990. 57 pages.

[Luqi86] Luqi. *Rapid prototyping for large software system design.* PhD thesis, University of Minnesota, 1986. 103 pages. .

[Luqi&88] Luqi & Katebchi, M. A computer aided prototyping system. *IEEE Software*, 66 – 72, 5, 2 1991.

[Luqi89] Luqi. Software evolution through rapid prototyping. *IEEE Software*, 13 – 25, 22, 5 1991.

[Mortensen90] B. Mortensen (Coordinating Proposer). *IPTES: Incremental Prototyping Technology for Embedded Real-Time Systems. Part II. Project Description.* Technical Report, IFAD, Odense, Denmark, January 8, 1990.

[Reilly&87] Reilly, E.L., Brackett, J.W. An experimental system for executing Real-Time Structured Analysis models. In , editor, *Proceedings of the XII Structure Methods Conference. Chicago, Illinois.*, pages 301–313, Chicago, Structured Techniques Association, May 1987.

[Royce90] Walker Royce. TRW's Ada Process Model for Incremental Development of Large Software Systems. *12th International Conference on Software Engineering*, ():2–11, 1990.

[TRW89] Ann Marmor-Squires. *Process Model for High Performance Trusted Systems in Ada.* Technical Report, TRW Systems Division, Fairfax, Virginia, August 1989.

[Ward&85] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems.* Yourdon Press, New York, 1985-86.

[Webb&86] Webb, M., Ward, P. Executable Data Flow Diagrams: An Experimental Implementation. In , editor, *Proceedings of the Structured Development Forum VIII. Chicago, Illinois.*, pages 1–21, Chicago, Structured Techniques Association, August 1986.