Chapter 17

PUSSEE METHOD IN PRACTICE

Specification and design of an embedded telecom system based on HIPERLAN/2 protocol

Nikolaos S. Voros
INTRACOM S.A., Patra, Greece

Abstract: The main goal of this chapter is to demonstrate the feasibility of PUSSEE development framework. For that purpose, a case study borrowed from the telecommunication domain is used in order to exhibit the applicability the method and associated tools for the design of complex systems. The application described is part of an embedded system based on HIPERLAN/2 protocol.

Key words: UML-B profile, embedded telecom systems, formal refinement, system decomposition.

1. AN OVERVIEW OF HIPERLAN/2 PROTOCOL

HIPERLAN/2 protocol provides data rates up to 54 Mbits/sec for short range (up to 150 m) communications in indoor and outdoor environments. Typical application environments are offices, homes, exhibition halls, airports, train stations and so on.

In order to specify a radio access network that can be used with a variety of core networks, the HIPERLAN/2 standard [1] provides a flexible architecture that defines core independent physical (PHY) and Data Link Control (DLC) layers and a set of convergence layers that facilitate access to various core networks including Ethernet, ATM and IEEE 1394 (Firewire).

The air interface is based on time division duplex (TDD) and dynamic time division multiple access (TDMA). It relies on cellular networking topology combined with ad-hoc networking capability, and supports two basic modes of operation: centralized mode (CM) and direct mode (DM). In

the CM operation every radio cell is controlled by an access point covering a certain geographical area, and mobile terminals communicate with one another or with the core network through the access point. In the DM operation, mobile terminals in a single cell network can exchange data directly with one another. The access point controls the assignment of radio resources to the mobile terminals. Figure 17-1 outlines the protocol architecture, while Figure 17-2 delineates the scope of HIPERLAN/2 standards.

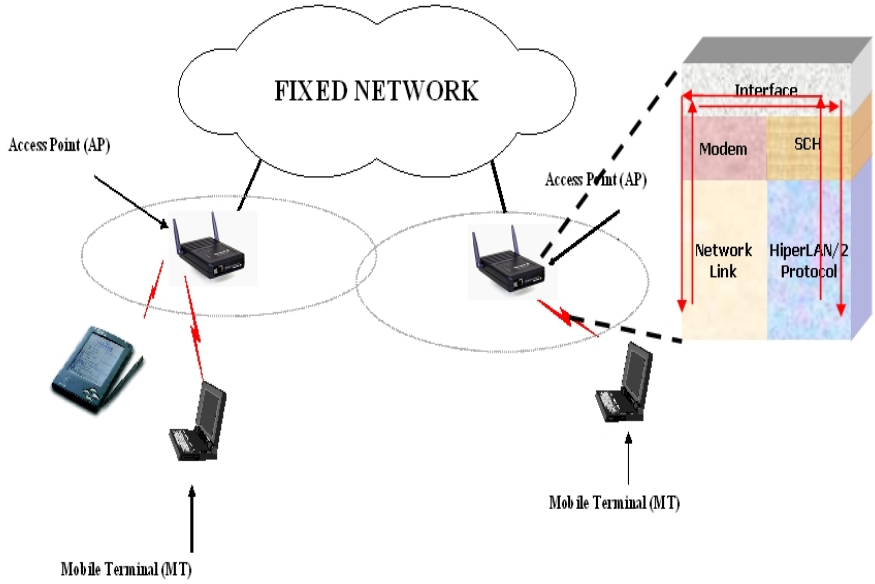

Figure 17-1. An overview of HIPERLAN/2 architecture

The system under design is part of the access point system and consists of the AP scheduler and the modem. The next paragraphs describe the design of the specific case study using the design steps supported by PUSSEE method.

In Figure 17-3 the final architecture of a prototype for the HIPERLAN/2 based system is described. The final system implementation employs both hardware (e.g. the HIPERLAN/2 modem), and software (e.g. DLC layer) components. Regarding the software part of the system, the design of the Frame Scheduler for the Access Point (AP) is presented. The latter, lies in the MAC sub layer of the DLC layer and is responsible for the design of MAC frames.

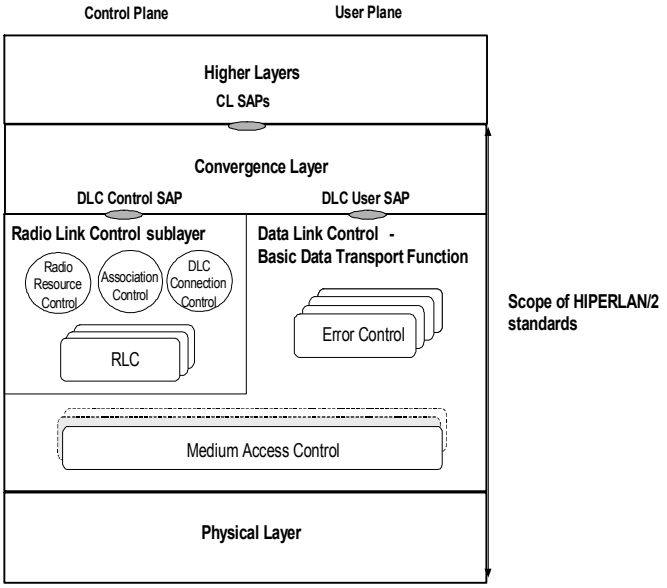

Figure 17-2. The scope of HIPERLAN/2 standards

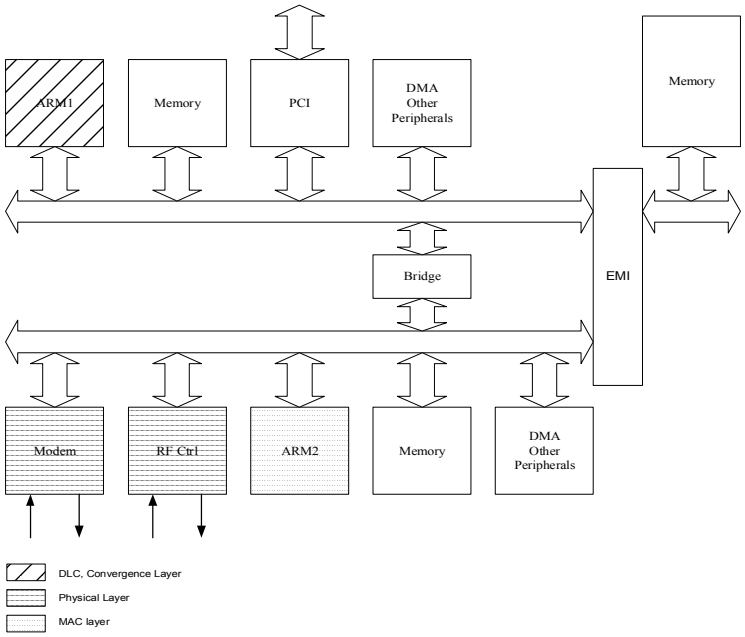

Figure 17-3. The HIPERLAN/2 prototype

2. SYSTEM SPECIFICATION USING UML-B PROFILE

In Figure 17-4 part of the overall system specification using UML-B profile [2] is presented, and corresponds to the SCH box (Access Point Sceduler) depicted in Figure 17-1. The main parts of the Access Point Scheduler [1] include:

- AP_SCHEDULER which is responsible for the design of a MAC frame.
- TRAFFIC_TABLE that describes the next frame's logical channel entries required, according to the resource requests.
- FRAME_INFO that decides the number of information elements (IEs) and the number of blocks required (each block contains three IEs, the number of idle IEs and the number of padding IEs).
- DECISION module that contains the decision algorithm used.
- FCH that contains the resource grants for the FCCH channel.

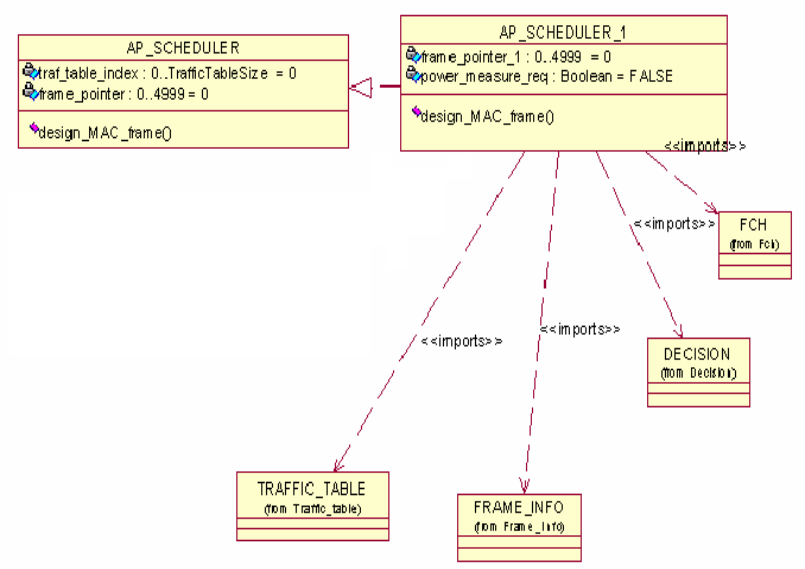

Figure 17-4. Description of HIPERLAN/2 Access Point Scheduler using UML-B profile

AP_SCHEDULER_1 is a formal refinement of the initial AP_SCHEDULER. The "imports" arrows refer to the corresponding keyword of B language and represent B modules containing part of AP_SCHEDULER functionality.

3. FORMALLY PROVEN MODEL REFINEMENT

For the design of the AP scheduler subsystem, the UML models of Figure 17-4 were translated to B machines using U2B translator [3]. The functionality of each class (including its attributes and operations) is thoroughly described in [4].

Figure 17-5 presents the B code produced by U2B translator for the AP_SCHEDULER class. AP_SCHEDULER_1 class was also translated to B using U2B, and the required proof obligations for the specific model refinement were generated and proven using Atelier B [5], as depicted in Figure 17-6.

```
MACHINE AP_SCHEDULER_CLASS
/*" U2B3.6.4 generated this component from Class AP_SCHEDULER "*/
CONSTANTS
        AP_SCHEDULER
PROPERTIES
        AP_SCHEDULER = 1..n
VARIABLES
        traf_table_index,
        frame_pointer
INVARIANT
        traf_table_index : AP_SCHEDULER --> 0..TrafficTableSize  &
        frame_pointer : AP_SCHEDULER --> 0..4999
INITIALISATION
        traf_table_index :: AP_SCHEDULER --> {0} ||
        frame_pointer :: AP_SCHEDULER --> {0}

OPERATIONS
 design_MAC_frame (thisAP_SCHEDULER) =
BEGIN
    traf_table_index, frame_pointer:(traf_table_index:0..TrafficTableSize &
         traf_table_index<=2+6*(NumberOfMTsPlusBroadcast-1) &
         frame_pointer:0..4999)
  END

END
```

Figure 17-5. The initial B code produced for the AP_SCHEDULER

During the refinement process, appropriate predicates were defined to express the properties of the linking (gluing) invariants between the initial B model and the refined B model. For example, the following invariant is defined in the B code of Figure 17-5:

```
traf_table_index: AP_SCEDULER --> 0..TrafficTableSize
```

which states that variable *traf_table_index* belongs in the range *0..TrafficTableSize*. The specific invariant generates the following proof obligations:

1. $traf\_table\_index \leq TrafficTableSize$
2. $0 \leq traf\_table\_index$

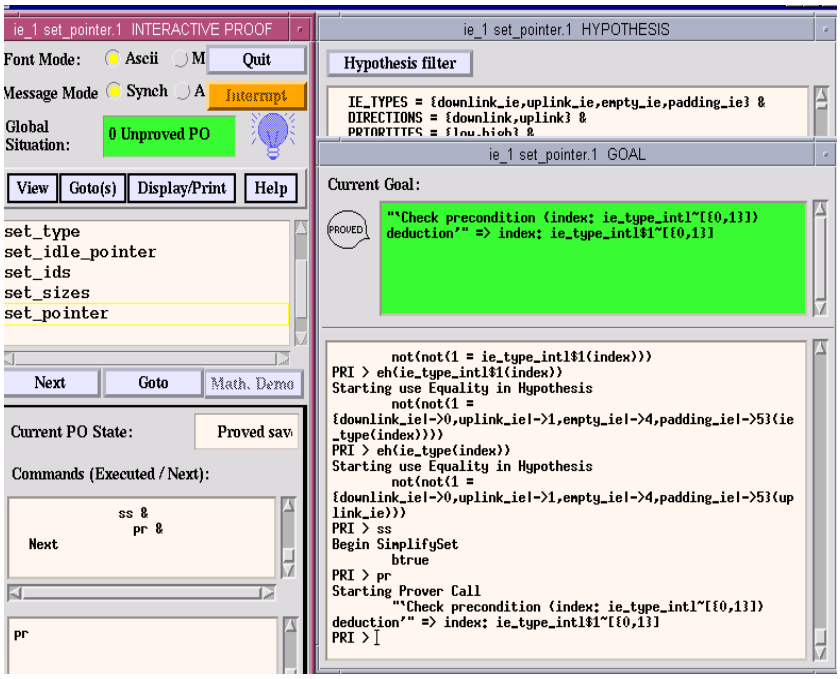

Figure 17-6. The interactive prover of Atelier B

The proof obligations generated were mainly proven using the automatic prover of Atelier B, but there were also cases where the designers had to prove several proof obligations using Atelier B's interactive prover. Moreover, due to the complexity of certain proof obligations, the designers experienced excessive numbers of proof obligations (sometimes more than 100), which were impossible to prove using the interactive prover. In that cases, there were two different alternatives to follow:

- Introduction of an additional intermediate refinement level between successive model refinements, to express some properties at an intermediate level of detail. In many cases this technique can improve the task of proving significantly, while the non proven proof obligations can be proven more easily at the lower refinement levels.
- Introduction of a new module (or more if necessary) to simplify the proving procedure. The new module can make simpler the complex operations (the ones that create excessive numbers of proof

obligations). The B machine of the new module is imported to the lower level refinement with the use of the IMPORTS clause, supported by B language.

In the case study presented, the first alternative was adopted, while in cases of excessive number of proof obligations new intermediate B modules were imported. The refinement process revealed 3.343 proof obligations; 3.106 of them (92,6%) were automatically proven using Atelier B's automatic prover, while 247 proof obligations (7,4%) were proven using the interactive prover of Atelier B.

4. SYSTEM DECOMPOSITION

System decomposition takes place using the decomposition assistant tool [6]. The tool intends to support system partitioning into hardware and software; it accepts formally proven system models described in eventB [7], and based on a profile like the one presented in Figure 17-7, produces subsystems along with the required communication interface. The subsystems and their interfaces can be further formally refined until a fully functional subsystem model is reached. In order to identify and prove the proof obligations required during subsystem refinement, Atelier B is used. The final subsystem implementation emerges through direct translation of the eventB code either to C or to VHDL. For the latter, the BHDL translator [8] developed in the context of PUSSEE Project [9] can be employed.

For the HIPERLAN/2 case study, the B models emerged from the previous design stages were used as input to the decomposition assistant tool in order to partition the system into hardware and software. Part of the system decomposition profile used is presented in Figure 17-7. It contains the number and the names of the subsystems specified by the designers, as well as variable allocation. Communication is deduced from that description, using default communication protocol for accessing data. These communication protocols are likely to be extended.

The system under design was decomposed in two subsystems:

- SS_SCH subsystem which corresponds to the functionality of the system UML-B model of Figure 17-4.
- SS_MODEM subsystem which contains the part of the initial system that contains the HIPERLAN/2 modem functionality.

```
THEORY DECOMPOSITION IS
        SS_SCH={Receive, Transmit, flush, assemble};
        SS_MODEM={IQ_EN, RESET, ENDOP, NOP, CFG, RX}
END
&
THEORY SYNCHRONISATION END
&
THEORY ALLOCATION IS
         Allocate(newi, SS_SCH);
         Allocate(received_pointer, SS_SCH);
         Allocate(source_address, SS_SCH);
         Allocate(destination_address, SS_SCH);
         Allocate(src_ptr, SS_SCH);
         Allocate(current_sar_length, SS_SCH);
         Allocate(dest_length, SS_SCH);
         Allocate(buffer, SS_SCH);
         Allocate(valid_items, SS_SCH);
         Allocate(cell_arrived, SS_SCH);
         Allocate(cell_sent, SS_SCH);
         Allocate(PHY_Mode, SS_MODEM);
         Allocate(Valid, SS_MODEM);
         Allocate(Slot_Num, SS_MODEM);
         Allocate(PDU_Type, SS_MODEM);
         Allocate(EndM, SS_MODEM);
         Allocate(RPT, SS_MODEM);
         Allocate(Datatype, SS_MODEM);
         Allocate(EndT, SS_MODEM);
         Allocate(Enable, SS_MODEM);
         Allocate(EndR, SS_MODEM);
         Allocate(FrameNumber, SS_MODEM)

END
```

Figure 17-7. The decomposition profile for the telecom case study

5. HARDWARE/SOFTWARE ALLOCATION & IMPLEMENTATION

The final step of the design process was the hardware/software allocation and the generation of final code. In the context of the HIPERLAN/2 case study, only the SS_SCH subsystem was implemented. For that purpose, 2291 lines of C code were generated using Atelier B's C code generator and the final C code has been tested on ARM7 TDMI.

6. PUSSEE METHOD EVALUATION

The next paragraphs describe the pros and cons of using the proposed method in the context of an industrial environment for the development of complex telecommunication products.

6.1 Methodology adoption

As already described in Chapter 3, the PUSSEE method relies on the combined use of UML and B. The UML specifications are written in a B compliant manner, using the UML-B profile proposed by the PUSSEE approach.

Even though the PUSSEE method appears to be compatible with the practices used by many telecom companies, the use of B language for proving system properties might be a barrier to the extensive use of the method. The experience gained from the design of HIPERLAN/2 system revealed that a strong mathematical background (especially in the domain of predicate calculus) is required for the engineers that plan to use the PUSSEE approach. As a result, the potential use of the PUSSEE method as part of an existing development process will definitely require training courses of the design teams in order to use productively the proposed approach. The cost/benefit ratio of the latter will definitely play a crucial role in the future adoption of PUSSEE method.

The potential use of PUSSEE method in practice should rely on a combination of the U2B and Atelier B tools. For Atelier B, a possible configuration would include:

- **Atelier B software:** Fully functional, basic configuration for one server including standard graphic user interface, syntax and type checker, proof obligations generator, multi-pass automatic prover and interactive prover with graphic user interface, documentation tools and translators for C and C++.
- **Training:** Training sessions for understanding fundamental principles of the B method (Level 1) and how to develop in B (Level 2).
- **On going support:** Support including, answering questions and determining the best way to efficiently introduce B in a real world design environment development process will be necessary.

The aforementioned requirements reflect the basic version of Atelier B and their cost is 45.000 Euros[1] [5]. Additionally, the configuration may also include:

[1] The prices reported are the official prices of ClearSy S.A., June 2004.

- **User rule proof tools** for validating the mathematical rules added by users during proof, at the cost of 6.000 Euros.
- **Training** for an additional design team at the cost of 15.000 Euros.
- **Four additional licenses** to supplement the basic license at the cost of 3.600 Euros.
- **Maintenance** of the basic tool version including bug fixes and product updates at the cost of 4.800 Euros.

Table 17-1 presents a summary of the total cost of PUSSEE method.

Table 17-1. Total cost of PUSSEE method tools and training

TOOL	COST
U2B translator	Free of charge
Atelier B: Basic configuration	45.000 Euros
User rule proof tools	6.000 Euros
Additional training	15.000 Euros
Additional licenses	3.600 Euros
Maintenance	4.800 Euros
Total cost	74.400 Euros

One additional issue that must also be taken into account is the fact that the method must be mature enough before adopted for the development of commercial products. In the context of a product line, maturity is close related to parameters like stability, on going support, adequate documentation and ability to handle highly complex system models. In its current version, the method is mostly supported by academic tools or prototypes that are still under development. In the context of the case study presented, the tools supporting PUSSEE methodology have proven their value. What remains is to see the combination of PUSSEE methodology and the supporting tools under a more robust development framework.

6.2 Methodology expressiveness

B language is traditionally used for the development of safety critical systems. Thus, in order to provide error free system models of the system under development there are several descriptions that must be imposed. The constraints can be divided into four main methodological notions of B developments:

- *Preservation of the local invariants*

  - The operations of a specific machine can be called only by one machine. This restriction prevents data sharing involving multiple write access.
  - Simultaneous operation calls are forbidden.
  - Each variable of a machine can be altered by, at most, one of the simultaneous substitutions of an operation.
- *Strict tree call structure*
  - Loops within the calling structure of a set of machines are not allowed.
  - Local operations cannot call other operations within the same machine.
- *Encapsulation principle*
  - A variable of a machine can only be written by the operations of the machine containing it.
- *No recursitivity of the operation calls*
  - Simultaneous operation calls are forbidden.

Even though the aforementioned restrictions are essential for the development of formally proven system models, there are cases (especially in the telecom domain) where they might be restrictive. For example, during the design of the AP scheduler a significant part of the scheduler had to be re-designed in order to be compliant with B language primitives. One additional reason that imposed redesigning system parts was the excessive number of proof obligations generated form the initial model. In general, the use of B language requires the definition of a significant number of system properties that must be expressed in the form of invariants. The latter, can lead to significant problems during the proving process, especially when we are dealing with complex systems.

6.3 Tool support

In the context of the HIPERLAN/2 case study, U2B translator, Atelier B and decomposition assistant have been mainly used. The experience gained from their use is described in the next paragraphs.

6.3.1 U2B translator

U2B is a tool, which through a flexible user interface allows translation of UML models (written using the UML-B profile) to B language. It is available in two flavors:

- *U2B3* (version 3) that relies on UML models created using Rational Rose (and the conventions adopted by Rational) and,

- *U2B4* (version 4), which is tool neutral and relies on XML. U2B translates the UML models first to XML language and then to B code. The use of U2B4 for the translation of the UML-B models of the AP scheduler has reveled several restrictions in the way the UML models should be constructed. Additional, there is an inherent difficulty to deal with complicated models.

Additional restrictions of the U2B tool come from the fact that B does not support cyclic structures. As a result, in order to make a B model, the designers had to produce tree-structured UML model. Moreover, during the construction of the UML models the designers should keep in mind that the system will be translated in B and thus using B definitions for variables and functions. If this is not the case, U2B will create a B model, but it will probably not pass the proofing process.

Based on the experience gained for the use of U2B tool, we could say it appears to be a promising tool, which could potentially bridge the gap between UML and B by isolating the designer from the B language details. As a result, designers that are not experts in using B language could use PUSSEE method and take advantage of its benefits. The latter presupposes that UML-B profile and the U2B can be used as a front end that isolates the designers from B language details as much as possible.

6.3.2 Atelier B

Atelier B was the main tool employed for the development of B models of HIPERLAN/2 case study. It was also used for generating and proving the required proof obligations between successive refinements. At the last phases of the development cycle, Atelier B was also used to produce C code for the software part of the final system.

From the total number of POs generated during the refinement process, 92,6% were automatically proven using Atelier B's automatic prover, while only 7,4% of them were proven interactively. Despite the high percentage of automatically proven POs, there were restrictions in Atelier B which are directly related to the nature of B language. In addition, due to the excessive number of proof obligations produced during the early design phases of system design, significant model restructuring was required. Despite the effectiveness of the proving process of Atelier B, there were also cases where the designers came across inefficiencies throughout the proving process. The rule base of Atelier B, in spite of the 2.200 rules it supports, should be enriched in future tool versions in order to ease the proving process. Significant problems were also experienced with proofs that involved cardinal numbers and Σ functions.

Interoperability among the tools is another significant parameter for the design of complex systems. Communication (e.g. model exchange) between

Atelier B with other tools like U2B would definitely be an advantage. Moreover, availability of the tool for different operating systems might be helpful towards this direction.

6.3.3 Decomposition assistant

Decomposition assistant aims at producing system partitions into hardware and software subsystems, based on formally proven system models. In the current version, it only accepts as input system models described in eventB. In the case study presented, this was a major problem since Atelier B relies on B language for generating and proving proof obligations. As a consequence, the designers had to use translators for translating eventB to B and vice versa [10]. This produced significant delays in the partitioning process since the two languages are not fully compatible.

Moreover, the user interface provided by the tool was not adequate, while significant support was required as far as the subsystem interfaces were concerned. What would be expected in forthcoming versions of the tool would be a library of formally proven standard interfaces that could be customized, and possible extended, according to the needs of each subsystem.

6.4 Final product

The last part of the HIPERLAN/2 case study was the generation of C code for the final implementation of the system. The code produced was based on the B implementations constituting the AP scheduler, and for its production Atelier B's automatic translator for C code has been employed. As a general remark we could mention that the code produced by Atelier B was well documented and easy to understand by the designers (the code produced was about 2291 lines in C). Although no optimization techniques were used, it would be preferable to have the ability to produce code for different implementation platforms e.g. for ARM7 TDMI.

In terms of productivity, the code production is a fairly easy process while the fact that system designers are able to produce C code form formally proven to be correct B implementations is definitely an advantage since in allows the detection of design flaws early enough in the design process.

7. SUMMARY

In the previous sections we presented an insight on how PUSSEE method could be employed in a real world design environment. The case study presented is based on HIPERLAN/2 protocol, and has been designed using PUSSEE method and the tools supporting it. For the initial system specification the UML-B profile has been adopted, while for system design B language/method and Atelier B have been used. The latter has been utilized in order to verify formally the correctness of the model refinements in B, as they emerge from the U2B translator.

Based on the experience gained form the design of the HIPERLAN/2 case study, an initial evaluation of PUSSEE method has been presented. In a nutshell, PUSSEE method appears to be a promising design approach, the benefits of which could be exploited in the context of a real world design environment. Nevertheless, there are issues that must be taken into account in future versions of the method, mainly related to the tool interoperability and their efficient use in an existing product development process.

REFERENCES

1. ETSI. 2000, *Broadband Radio Access Networks BRAN; HIPERLAN Type 2; Data Link Control (DLC) Layer Part1: Basic Data Transport Functions*, ETSI TS 101 761-1 v1.1.1.
2. C. Snook, M. Walden, *Use of U2B for Specifying B Action Systems*, Proceedings of International Workshop on Refinement of Critical Systems: Methods, Tools and Developments, Grenoble, France, January 2002.
3. C. Snook, M. Butler, *U2B Downloads*, Available at: http://www.ecs.soton.ac.uk/~cfs/U2Bdownloads.htm.
4. K. Antonis, N. Voros, *D3.1.2: Specification of the Telecom System-on-Chip Experiment*, IST-2000-30103 PUSSEE, Project Report, 2002.
5. Atelier B, Available at: http://www.Atelier B.societe.com/, 2003.
6. T. Lecomte, *D4.4.1: Methodological Guidelines: Interface based synthesis/ refinement in B*, IST-2000-30103 PUSSEE, Project Report, 2003.
7. ClearSy, *Event B Reference Manual v1.0,* Available at: http://www.atelierb.societe.com/ressources/evt2b/eventb_referencemanual.pdf, 2001.
8. KessDA, *BHDL User Guide Preliminary Version*, Available at: http://www.keesda.com/pussee/bibliography.htm
9. PUSSEE Project, Available at: http://www.keesda.com/pussee, 2003.
10. MATISSE Project, *Event B to B Translator User Manual*, IST-1999-11435, Project Report 1999.