Adaptive, Safe Mobile Robot Programming in the Intelligent Space

Laszlo A. Jeni†, Zoltan Istenes‡, Mate Tejfel‡, Peter Korondi*, Hideki Hashimoto†
†University of Tokyo, Institute of Industrial Science, ‡Eotvos Lorand University, Dep. of Software Technology and Methodology, * Budapest University of Technology and Economics, Dep. of Automation and Applied Informatics

Abstract — **This paper describes how the safe mobile code technology can be integrated into the Intelligent Space environment. In the Intelligent Space, several Distributed Intelligent Network Devices communicate and share their information about a human environment. In this environment mobile robots can be controlled with mobile code technology. The mobile code is a program-component obtained from a remote system, transferred across a network and dynamically downloaded and executed on the robots. This code is created, verified, stored and transmitted to the robot using the Certified Proved-Property-Carrying Code architecture, where properties and their proofs also attached to the code. The receiver can verify the proofs and it can decide whether to use or to refuse the received mobile component. Information about the robot's environment is also sent to the robot from the Intelligent Space. Robots contain explicit and formally expressed security requirements. Explicit and formal properties of the mobile code are attached to the mobile code. Then a formal verification system can verify the mobile code properties correspondence against the robots requirements. The robot refuses to execute those mobile code tasks violating it's requirements.**

Keywords — **Intelligent Space, distributed sensors, CPPCC, mobile code, mobile robots, requirement verification.**

I. Introduction

THE field of robotics is closely related to Artificial Intelligence. Intelligence is required for robots to be able to handle such tasks as navigation, referred to as robotic mapping including the sub-problems of localization (knowing where you are), mapping (learning what is around you) [1] and path planning (figuring out how to get there) and such as manipulate objects (usually described in terms of configuration space).

Robots surronds us more and more. They work around us and they help us. The human environment is much more complex and complicated for the robots than a "laboratory" or a "research" environment. Robots need lots of information about the human environment to be able to achieve their tasks.

This paper illustrates a model of a system using mobile robots having the following three main goals.

1. The robot works in a human environment.
2. The robot executes various tasks.
3. The robot is safe, in that sense, that the actions of the robot are formally verified against a set of security requirements.

The system contains multiple robots, the robots cooperate with each other and with other components of the system. To achieve the previous three goals the system needs the following three requirements.

1. The system needs information from the human environment.
2. The system uses mobile code which can be downloaded and linked dynamically, with the description of the various tasks.
3. The system uses formal verification system which is able to check the correspondence of the descriptions of the formal tasks against the security requirements of a robot.

There exists several models to realise these three requirements separately [2]. Our model integrates all of them in a coherent system using the following three components.

1. Intelligent Space described in Section 2.
2. Mobile code technology illustrated in Section 3.
3. CPPCC architecture and the B-method both presented in Section 4.

Section 5 illustrates the integration of these components mentioned above. The final section contains the conclusion.

II. Intelligent Space

The Intelligent Space (iSpace) is a space (room or corridor), which has distributed sensory intelligence (various sensors, such as cameras and microphones with intelligence) and actuators (TV projectors, speakers, and mobile agents) to manipulate the space [3]. For the illustration of the main concept of the iSpace see Fig. 1.

A space becomes intelligent, when Distributed Intelligent Network Devices (DINDs) are installed in it [4]. A DIND has a sensing function through devices such as a camera and microphone that are networked to process the information in the Intelligent Space. The DINDs monitor the space, achieve data and share them through the network [5].

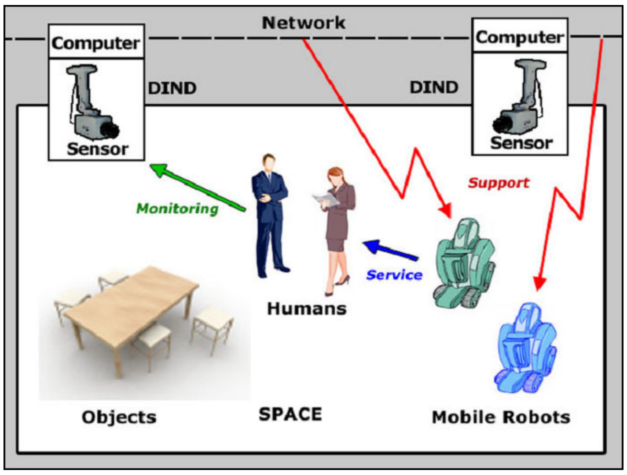

Fig. 1. The iSpace concept.

The iSpace also consists of humans and not only of sensors, cameras or robots. For instance, the iSpace can recognise humans, track their movement to identify the walking areas and learn the shortest safest path in the environment [6][7].

The iSpace is a system for supporting people in it. Events, which happen in it, are understood. However, to support people physically, the intelligent space needs robots to handle real objects. Mobile robots become physical agents of the Intelligent Space and they execute tasks in the physical domain to support people in the space. Moreover, robots can understand the requests (e.g. gestures) from people more effectively. The applicable tasks include movement of objects, providing help, for example to aged or disabled persons etc [8].

The ongoing research activities about Intelligent Space achieved several results and solutions in the field of motion control [9], feature extraction [10], recognition and tracking the path of moving objects [11]. Recent research focuses on image recognition and on solutions that are developed on the analogy of the human vision processing [12].

III. Controlling the Robots Using Mobile Code Technology

Several different technologies can be applied to control robots. We can distinguish three main approaches of controlling.

The most basic method is called teleoperation. The robot sensor data is transmitted to a remote operator. The operator sends each command separately back to the robot. These commands are very simple and they command the robot "step by step", such as turn left, go ahead, grab, etc. This method requires a continuous and reliable connection with the robot. For example this method is used for remote controlled robotic arms.

In the second method the robot controlling program is running directly on the robot. The robot can execute only one program which is initially installed on it. The robot executes the program autonomously and it can only receive control data, for example, checkpoints it has to attain. In this method while the robot executes his control program it accesses its sensor data, makes decisions and controls its actuators and motors. The advantages of this method is the autonomy of the robot and the small need of communication. Still it also has disadvantages. The robot controlling program can be adequate for some situations, but if the robot environment changes, the robot control program easily become outdated and not well suited to the changed situation.

The third method tries to overcome the problems of the second method. The robot executes a framework which downloads and executes the robot controlling program dynamically. The dynamically downloaded program can be referred as "mobile code". Mobile code is a program or a program-component obtained from a remote system, transferred across a network and dynamically downloaded and executed on a local system [13]. Examples of mobile code include plugins, JavaScripts and ActiveX controls (executed in web browsers) or codecs (executed in multimedia players). The mobile code technology is widely used in fields with multi-purpose and dynamically determined execution. This method does not require continuous connection, but still provides a high flexibility in the execution of the various tasks.

From these technologies our model uses the mobile code technology. One of our goals is to support the flexibility of the use of robots. More precisely we would like to make the robots capable to execute various tasks which are not known in advance and which are assigned to the robots "on the fly", dynamically by the iSpace. The mobile code technology is well suitable to achieve this goal.

In our model the robots are parts of the iSpace, they cooperate with each other and with other parts of the iSpace and they also share their information. This architecture does not contain central controller. Humans and DINDs of the iSpace can send mobile codes to the robots. During the execution of their tasks robots can communicate with the iSpace and receive information about the environments.

The simplest way to assign the task to a robot is, when a human operator allocates the mobile code realising the task for a specific robot. The iSpace sends the mobile code to the selected robot who obtains it and then executes it.

Another possible way is, when a human operator only determines the mobile code and the iSpace searches for an adequate robot and allocates the robot to the task. This search can depend on different criteria, such as the availability, the characteristics and the position of the robot in the space, etc.

IV. Verification of the Requirements

Since the mobile robots exist, work, navigate and operate in human environment, it is natural to request them to satisfy certain basic requirements. In most of the cases these requirements exist only implicitly in the robot controlling code. For example the robot stops when its ultrasound distance sensor detects an object too close to it. In our model we would like to express these requirements explicitly, and verify whether the mobile code satisfies these requirements.

The requirements and the properties of the mobile codes are expressed in a formal way which allows the use of formal verification systems. This section presents the way of the formal definition of the requirements, then the used formal verification method and finally an architecture support to the verification and to the safe transmission of the mobile code.

These examples in this paper use the B-method [15] for the formal representation and the formal verification system, but the model could also use other formal representation and corresponding verification systems too. (For more detail see [14].)

The development process proceeds with the application of refinement steps until an "implementation" is produced. The refinement of a specification involves the reformulation of an abstract machine by making it more concrete and extending it with further details. The data structures and substitutions of the implementation are B constructs, independent of specific programming languages, but they can easily be translated into constructs of a concrete programming language. This code generation phase can produce source code in several widely used programming languages, for example Ada, Java and C#.

The process of proving the correctness of a program in B is structured according to the applied refinement steps. During each refinement it is proved that the properties of the original abstract machine follow the properties of the resulting (either abstract machine or implementation) unit. So the correctness of the implementation with its respects to its specified properties will be proven in this way. The whole proving process can be realised with the aid of automatic and semi-automatic proof tools.

A. *Requirements*

In the introduced model a robot is part of a complex system, the iSpace. It has to attend different demands of multiple entities (people, other DINDs of the iSpace). These demands are different tasks (missions) expressed as mobile code components assigned to the robot. The robot has certain requirements and it can refuse tasks which do not satisfy these requirements.

The requirements of the robot are formally defined properties (mainly invariants). For example these formal requirements can be the following.

- The robot may not go to places from where it is not able to go back to its service station.
- The robot is prohibited to go to closer than 5cm to any objects or people.
- The robot has some resource bounds (memory, time, power consumption etc.).
- There are prohibited places in the space (lift, stairway, dangerous places etc.).

Figure 2 illustrates an example using the B syntax. The example is a small part of a specification of a robot navigation system.

```
MACHINE
   robot_navigation
INCLUDES
   elemental_routines
SETS
   STATES = {Moving, Stopped};
   RETURN = {Success, Failure}
VARIABLES
   state
INVARIANT
   state : STATES & ( state=Stopped => x=0 & y=0 & dir = North )
INITIALISATION
   state:= Stopped
OPERATIONS
   r <-- route =
      PRE
         x=0 & y=0 & dir = North
      THEN
         state:=Stopped || r :: RETURN
      END
END
```

Fig. 2. The requirements imposed against the tasks.

In this example the variable `state` describes the current state of the robot (`Moving` or `Stopped`), the variables `x` and `y` define the current position of the robot and the variable `dir` describes the current direction of the robot. The robot has one requirement, that each task has to be started from, and stop in a given state. We can formulate this requirement as an invariant `state=Stopped => x=0 & y=0 & dir = North`, namely if the state of the robot is `Stopped` it has to be the place `x=0` and `y=0`, and its direction has to be `North`.

In our model the illustrated verification process and the safe transmission of the mobile codes are supported by a Certified Proved-Property-Carrying Code architecture. This architecture is described in detail in the following subsection.

```
r <-- route =
VAR
   n, freeAhead
IN
   state := Moving;
   n := 0;
   freeAhead <-- FreeAhead;
   WHILE (freeAhead = 1) & n < world
   DO
      Forward;
      n := n+1;
      freeAhead <-- FreeAhead
   INVARIANT
      dir = North & x = 0 & n = y &
      y >= 0 & y <= world & y : NATURAL & y:(-world)..world &
      !i.(i:INTEGER & i>=0 & i<n => ENV(0,i,dir) = 1) &
      freeAhead = ENV(0,n,dir)
   VARIANT
      world - n
   END;
   IF n=world
      THEN r := Success
      ELSE r := Failure
   END;
   TurnBack;
   WHILE n > 0
   DO
      Forward;
      n := n - 1
   INVARIANT
      dir = South & x = 0 & n = y &
      y >= 0 & y <= world & y : NATURAL & y:(-world)..world &
      !i.(i:INTEGER & i>=0 & i<n => ENV(0,i,North) = 1) &
      !i.(i:INTEGER & i>=0 & i<n => ENV(0,i+1,South) = 1)
   VARIANT
      n
   END;
   TurnBack;
   state:=Stopped
END
```

Fig. 3. A fragment of the implementation of the code.

B. *Formal verification of the mobile codes against the requirements*

The explicitly and formally expressed requirements are verified against the properties of the mobile code. The properties of the mobile code are expressed also explicitly, in the same formalism, with the use of the B syntax. Using the B-method the properties of the code are available in an explicit manner.

In our model the method to create the code and the properties is irrelevant. As a proof of concept, a simple mission program has been developed using the B-method (see Figure 3).

This mission tries to move the robot forward to the northern border of the world: it succeeds if the robot finds no obstacles in the way. After reaching the northern border or finding an obstacle, the robot is turned back and returns to the original position as required.

C. *The CPPCC architecture*

As we mentioned earlier in our model mobile code technologies are used to control some functionalities of a robot. The control code of a robot can be extended with components by dynamically linking the obtained piece of code to the application. Such technologies are extremely vulnerable against malicious codes, as well as against accidentally erroneous or improper code, especially because these mobile components can also be created by a second party. For this reason it is extremely important to verify the correctness of the components and to ensure a safe manner of their transmission.

This safe transmission can be done, for example, with the use of the proof-carrying code technique [16]. Proof-carrying code (PCC) is a piece of mobile code with attached properties and their proofs. The receiver of the code can verify the attached proofs, and – by investigating the attached properties – it can decide whether to use or to refuse the received mobile component. This technique has also some drawbacks. For example the verification of the proofs may significantly slow down the code receiver application. It is even possible that the code receiver does not have the necessary resources (memory, network bandwidth, CPU-time, proof-checking software) for verifying the proofs.

In our model an other technique is used to eliminate these drawbacks.

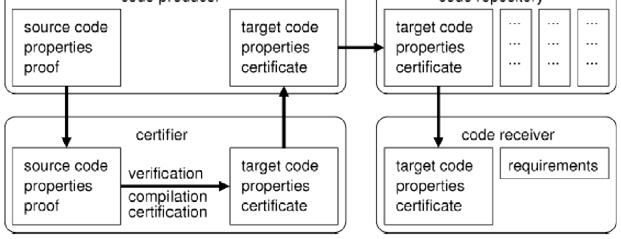

Fig. 4. Overview of the Certified Proved – Property - Carrying Code technology.

This technique is called Certified Proved-Property-Carrying Code [17] (or CPPCC, for short). CPPCC combines the certificate-based and the PCC approaches in order to provide highly efficient use of verified mobile program components. In this architecture the code receiver can make a decision on whether or not to accept and utilise the received code based on the declared properties of the code and on the opinion of a third-party, creditable certificate authority, which has verified whether the code had or had not the declared properties.

As Figure 4 illustrates, there are four different participants in a CPPCC system, the code producer, the certificate authority, the code repository and the code receiver. The scenario for producing and receiving safe mobile components is the following.

1. The code producer creates a program component and additionally it formulates and proves the properties of the created component based on the source code of the component. (Different kind of proof systems can be used for this purpose, ranging from manual theorem provers to fully automatic model checkers.) Finally the producer packs together the source code, the properties and the proofs and sends the package to the certificate authority.
2. The certificate authority checks that using the received proofs, the specified properties can be proved for the source code of the received program component. Then it creates the target code from the source code, packs the target code and the properties together, signs the package and sends it back to the code producer.
3. The code producer uploads the signed package to a code repository.
4. The code receiver obtains the mobile code from the code repository and checks the certificate attached to the received package. If the signer is trusted, it verifies whether the properties of the code match its requirements. If they match, the received code is linked into the code receiver and gets executed. If the certificate is not correct or the properties do not match the receiver refuses the execution of the code.

The first three steps are performed in static time with respect to the code receiver application.

V. Integration of the safe mobile code into the iSpace

In this section we present how the CPPCC architecture components are integrated in the iSpace environment.

In our model, the code producer can be DIND or a third party outside of the iSpace. When the producer of the mobile code is a DIND of the iSpace, the role of the certifier is not essential, since we can trust that the properties packed together with the code are really accomplished by the code.

```
The robot checks the certificate.
IF {the certificate is correct} THEN
   The robot obtains information from the
     iSpace.
   The robot combines the properties of the
     mobile code and the obtained information
     and compares them with its requirements.
   IF {the requirements match against the
       properties combined with the
       information} THEN
     The robot accepts and executes the code.
   ELSE
     The robot refuses the execution.
   END IF
ELSE
   The robot refuses the execution.
END IF
```

Fig. 5. The linker algorithm of a mobile robot.

But the verification whether the requirements of the robots match against these properties is essential, hence when the code is generated, the current state of the robot executing the code is not known (for example its position, the amount of its power, the amount of its free memory).

The second possible solution is when the code producer is not a part of the iSpace. In this case the certification and the verification process are both essential, since the producer is not trusted.

From the point of view of our model, it is not essential, whether the certifier is a part of the iSpace or not, only the fact whether we can trust it or not is important. The certifier can be a special DIND which does not require sensors or actuators and has only interface layer and local intelligence layer.

The code repository becomes part of the iSpace. The mobile codes describing the various tasks, are stored in the code repository.

As the result of some interactions of a human or of a DIND, the mobile code will be sent from the repository to the receiver. The repository can also be considered as a DIND. It can have low level physical devices and sensors to recognise the mentioned interactions (for example keyboard, camera, etc.), it also has some local intelligence to process the interactions and has some interfaces to get the mobile code from the code producer, to recognise the interaction of other DINDs and to send the mobile code to a given DIND, the code receiver.

The role of the code receiver will be played by a mobile robot in the iSpace. It obtains the mobile code containing the target code, the properties of the code and a certificate. After the robot has received the mobile code, a simple algorithm illustrated in Figure 5 is executed.

Comparing with the algorithm of the code receiver component of CPPCC illustrated in section 4 a little modification can be detected here. Namely, the requirements of the receiver (the robot) are compared not against the properties of the code itself but against a combined version of it. The reason for it, is that robot does not have global information about its environment. The information is stored in a scattered form in the iSpace, so the robot needs cooperation with other components of the iSpace to be able to deal with the properties.

```
MACHINE
   elemental_routines
SETS
   DIRECTIONS = {North, East, South, West}
VARIABLES
   x, y, dir
ABSTRACT_CONSTANTS
   ENV
CONSTANTS
   world
PROPERTIES
   world : NATURAL & world = 5 &
   ENV : (-world..world) * (-world..world) * DIRECTIONS -->  0..1 &
   ENV(1,0,West)=1 & ENV(0,1,South)=1 &
   ENV(-1,0,East)=1 & ENV(0,-1,North)=1 &
   !(i,j).(i:-world..world & j:-world..world-1 & ENV(i,j,North)=1 =>
                                                ENV(i,j+1,South)=1) &
   !(i,j).(i:-world..world & j:-world+1..world & ENV(i,j,South)=1 =>
                                                ENV(i,j-1,North)=1) &
   !(i,j).(i:-world..world-1 & j:-world..world & ENV(i,j,East)=1  =>
                                                ENV(i+1,j,West)=1)  &
   !(i,j).(i:-world+1..world & j:-world..world & ENV(i,j,West)=1  =>
                                                ENV(i-1,j,East)=1)
INVARIANT
   x: INTEGER & x : -world..world & y: INTEGER & y : -world..world &
   dir : DIRECTIONS
INITIALISATION
   x := 0 || y := 0 || dir := North
OPERATIONS
   TurnRight = CASE dir OF EITHER North THEN dir := East
                           OR     East  THEN dir := South
                           OR     South THEN dir := West
                           ELSE                dir := North
                           END
               END;
   ...
END
```

Fig. 6. The specification of the elemental programs of the robot.

For example, if the robot obtains the task to go to a given position placed in the space, it needs information about that position (is there any objects or humans, is it a dangerous place or part of a lift, etc.) to be able to determine whether this task hurts its requirements or does not.

The information about the space can also be described by using the B syntax. Figure 6 specifies that the world is of bounded size (`world = 5`) and that it is static: during the execution of a mission no new obstacles between grid positions appear (the `ENV` is `ABSTRACT_CONSTANTS`). The `ENV` function assigns the value of 0 or 1 to the x, y coordinates and direction triplets. For example `ENV(i,j,south) = 1` means that at the position `(i,j)` in the direction `South`, then there is a free way (no wall). The `PROPERTIES` clause defines that, obstacles are symmetric, namely there is an obstacle between A and B exactly when there is an obstacle between B and A (this is expressed by the last four properties of ENV).

Our model is the result of the described integration. It contains all flavours of the iSpace and additionally thanks to the CPPCC architecture, it provides the safe use of mobile robots and thanks to the adaptation of mobile code technology it also supports the flexible applicability of the mobile robots.

VI. Conclusion

This paper presents a model to provide a flexible and safe use of mobile robots operating in an iSpace. The model supports the flexibility by using mobile code technologies, describing the different tasks to be executed by the robots. To guarantee the safe execution of the code, the correspondence of the formal requirements of the robots is verified against the properties of the mobile code. In the current state of the work, we test several, different, possible implementations of the presented model.

Concerning future work, two main open questions could be mentioned. One of them would be, how to generate automatically a mobile code, how to obtain the (mobile) code and it's properties. Using the B method is and adequate tool to obtain the mobile code and it's properties, since, first an abstract specification of the problem is created, then this specification is refined into a concrete implementation, then the target code can be generated automatically [18] [19]. To generate automatically the mobile code, the code can also be assembled by combining smaller pieces of codes and their corresponding properties [20].

The second question could be the verification of the mobile code properties against the robot's requirements. If the code properties and the requirements are expressed in the same language, at the same level of abstraction (as in our examples) the verification is much easier than if they are expressed in different languages and at different levels of abstraction. In the latter case, if the requirements are very general and the properties are very detailed, then proving their correspondence can be very complex and can require a considerable amount of time [21].

At first sight, artificial intelligence and formal methods seems to be very far fields from each other in the informatics. In our model, robots use traditional artificial intelligence techniques and algorithms to recognise objects, to plan their paths, etc. The properties "attached" to these tasks allow formal verification systems to verify the safety requirements. This could guarantee a safer use of robots in our real, human world's environment.

References

[1] Z. Vamossy: Map Building and Localization of a Robot Using Omnidirectional Image Sequences. In: Proc. 4th International Symposium on Applied Computational Intelligence and Informatics (SACI 2007), Timisoara, Romania, 2007, pp. 191-194.

[2] Z. Istenes, T. Kozsik: Commanding a robot in a safe way. In: Proceedings of the 10th Symposium on Programming Languages and Software Tools (SPLST 2007), Budapest, Hungary, 2007, pp. 167-177.

[3] P. Korondi, H. Hashimoto: Intelligent Space, as an Integrated Intelligent System. Keynote paper of International Conference on Electrical Drives and Power Electronics, Proceedings, 2003, pp. 24-31.

[4] H. Hashimoto: Intelligent Space -How to Make Space Intelligent by Using DIND-. In: Proceedings of the 2004 TRS Conference on Robotics and Industrial Technology, Thailand, 2004, pp.1-11

[5] J. Lee, K. Morioka, N. Ando, H. Hashimoto: Cooperation of Distributed Intelligent Sensors in Intelligent Environment. IEEE/ASME Transactions on Mechatronics, Vol.9, No.3, 2004, pp.535-543, ISSN 1083-4435

[6] P. T. Szemes, H. Hashimoto: Estimation of Walking Habit in iSpace. In: Proceeding of the 4th International Symposium on Advanced Intelligent Systems (ISIS 2003), 2003, pp.531-534, Jeju, Korea, ISSN 1738-0073

[7] L. A. Jeni, Z. Istenes, P. Korondi, H. Hashimoto: Hierarchical Reinforcement Learning for Mobile Robot Navigation using the iSpace Concept. In: Proceeding of 11th IEEE International Conference on Intelligent Engineering Systems (INES 2007), Budapest, Hungary, 2007

[8] P. T. Szemes, J. Lee, H. Hashimoto, P. Korondi: Guiding and Communication Assistant for Disabled in Intelligent Urban Environment. In: Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechtronics (AIM), Kobe, Japan, 2003, pp.598-603.

[9] P. T. Szemes: Human Observation-based Motion Control Strategies in Intelligent Space. PhD Thesis, Tokyo University, Tokyo, 2005

[10] K. Morioka, H. Hashimoto: Color Appearance Based Object Identification in Intelligent Space. In: Proc. of the 8th IEEE International Workshop on Advanced Motion Control, Kawasaki, Japan, 2004, pp. 505-510

[11] B. Resko, P. T. Szemes, P. Korondi, P. Baranyi, H. Hashimoto: Artificial Neural Network based Object Tracking. In: Proceedings of SICE Conference, Sapporo, Japan, 2004, pp. 1398-1403.

[12] Z. Petres, B. Resko, P. Baranyi, H. Hashimoto: Biology inspired intelligent contouring vision device in intelligent space. In: Proc. of the 6th International Symposium on Advanced Intelligent Systems, Yeosu, Korea, 2005, pp. 865-870

[13] C. Ghezzi, G. Vigna: Mobile Codes Paradigms and Technologies: A Case Study. In: Proceedings of the 19th International Conference on Software Engineering, LNCS 1219, Springer-Verlag, Berlin, Germany, 1997

[14] Z. Istenes, T. Kozsik, Cs. Hoch, L. A. Toth: Proving the correctness of mobile Java code. In: Proc. of 6th Joint Conf. on Math. and Comp. Sci., Pecs, Hungary, accepted to Pure Mathematics and Applications, SAAS Ltd.-SAAS Publishing, Budapest, Hungary, Vol. 17 (2006), No. 34,.

[15] J. R. Abrial: The B-Book. Cambridge University Press, 1996

[16] G. Necula: Proof-carrying code. In: Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France, 1997, pp. 106-119

[17] Z. Horvath, T. Kozsik: Safe mobile code CPPCC: Certified Proved-Property-Carrying Code. G. Czajkowski and J. Vitek, Resource Management for Safe Languages (in: ECOOP 2002 Workshop Reader, LNCS 2548/2002, Springer-Verlag), 2002, pp. 8-10.

[18] Rodin project http://rodin.cs.ncl.ac.uk

[19] Rodin platform http://www.event-b.org

[20] L. Lamport, M. Abadi: Composing Specifications. In: ACM Transactions on Programming Languages and Systems 15, 1, 73-132, 1993.

[21] L. Lovei, M. Tejfel, M. Meszaros, Z. Horvath, T. Kozsik: Comparing Specification with Proved Properties of Clean Dynamics. Conference of PhD students in Computer Science, Volume of extended abstracts, p. 71, 2006.