COMPILER OF AGENT HIGH LEVEL LANGUAGE

Róbert Kalmár
Master Degree Programme (2), FIT BUT
E-mail: xkalma01@stud.fit.vutbr.cz

Supervised by: František Zbořil Jr.
E-mail: zborilf@fit.vutbr.cz

Abstract: This paper present new agent based programing languages for wireless sensor networks. A low level language ALLL and a high level one called AHLL. There are explained the main concepts of implementation a compiler from AHLL language to ALLL like intermediate code, optimization and target code generation.

Keywords: ALLL, AHLL, Agent Low Level Language, Agent High Level Language, intermediate code, compiler, programing language

1 INTRODUCTION

The aim of this paper is to present an implementation of a compiler from AHLL to ALLL language. We will shortly present both languages, their purpose and some key features. In the following parts we emphasise on main concepts used by implementing a compiler, design of intermediate code representation used by this compiler and some techniques for implementing cycles and conditionals in ALLL.

At this time we have designed AHLL language and implemented a complete compiler in Java from AHLL to ALLL, but without any optimizations.

2 ALLL

ALLL [1] language was developed on FIT BUT. ALLL is an acronym from Agent Low Level Language by which agent's behaviour is controlled. This language [1] is designed to be as small as possible due to minimal size of device's dynamic memory, easily interpretable and the expressiveness should correspond to today's modern approaches to agent development as communication, reactivity to changes, hierarchy of plans etc.

ALLL's basic structure are plans. Plan is a sequence of actions in form of linear list enclosed in brackets. Possible actions in ALLL are manipulation with knowledge base and plan base (addition and deletion), direct or indirect plan execution, queries on knowledge base, platform service execution, changing active register and communication with other agents (sending and receiving messages).

By indirect plan execution plans are stored in plan base. By platform service execution the list contains name and parameters of the service. More information about ALLL can be found in [1, 4, 5]

3 AHLL

AHLL or Agent High Level Language compared to ALLL is designed to be more user friendly. It was developed also for programing agents in WSN. The purpose wasn't to design a completely new language but more to pick an existing commonly known one, slightly redesign it for our purpose and write a compiler from this language to ALLL.

AHLL is imperative [3, p. 41], block structured programing language. The program in AHLL consist of plans that may have several parameters. They are defined with a keyword `plan` and the definition have the form like definition of functions. The entry point of the program is defined by keyword `main`. This special plan cannot have parameters.

AHLL support conditional statements `if-then` and `if-then-else`. From loops AHLL support well known `while` statement and `foreach` statement, for iterating over a sequence. Like most of imperative languages AHLL allows usage of variables and evaluation of expressions.

4 REPRESENTING AHLL CONSTRUCTIONS IN ALLL

Just before we present a compiler implementation, we shortly show, how can be several structures from AHLL represented in ALLL.

Firstly we start with representing variables. Variables in ALLL can be represented as a sequence in form `(name,value)`. Or we can use a special platform service for storing variables.

Conditional statement can be represented as action for evaluating the condition followed by sequence of direct plan execution. In such a plans the first action is query on knowledge base for particular condition. Only one plan will succeed, others will fail on this query.

Loops [4] are represented similarly but the plan containing condition evaluation and loop body is stored in plan base because of the next iterations. The last action in this plan is indirect plan execution of the same plan. Once the condition stops to be true this plan fails and no more iteration will be executed.

5 COMPILER IMPLEMENTATION

Compilation process in common consists of few analysis of input source code, namely lexical, syntactical and semantic, to determine program structure. There are plenty of generators to automatic construction of lexers and parsers from EBNF representation, like Lex, Bison, ANTLR and much more. In our case we used ANTLR tool for specifying AHLL language grammar and generating lexical and syntactical analyzer of AHLL. The output of these analysis is Abstract Syntax Tree (AST), which is the first form of intermediate representation of program by compilation process.

Semantic analyzer was written by us and it use recursive descent on the generated AST and transforms AST into next intermediate representation of program called MIR code. It uses symbol stack structure to store symbols and constants appeared in program. Stack structure is used because of variable overlapping in scopes (plans, blocks etc.). Plan table and service table are used for storing information about defined plans and services. There is also a symbol table structure, which contains references for every symbol and constant defined in compiled program, but is mostly for debugging purpose because MIR instructions that operate with symbols contains this references also.

6 INTERMEDIATE CODE

By the translation process the code is firstly transformed in its intermediate representation, usually called intermediate code. The translation into an intermediate code is useful mostly for optimization techniques which increases the program's performance. There are several intermediate representations of codes. According to theirs level we can divide [2] them to High- (HIR),Medium- (MIR) and Low-level (LIR) intermediate code. Examples of HIRs are Abstract syntax trees, MIRs 3-address codes and LIRs some forms of abstract representation of machine instructions.

In 3AK conditions and loops [2] are represented via condition evaluation and `GOTO label` instruction. But in ALLL there is no goto action, only sequences of actions and plans that can fail or succeed.

That is the reason we turned the conditional code in a form more as sequences and without `GOTO`. Example of representing conditions:

MIR pseudo-code	MIR example	Translation into ALLL
`if condition ID`	`if a > 5 ID=1`	`@( #(rel,(g,&1,5)),`
`  then-actions`	`send "ag1" "Hello"`	`  !(ag1,Hello)`
`[else ID]`	`else ID=1`	`)@( #(rel,(le,&1,5)),`
`  else-actions`	`a <- a + 1`	`  $1,#(ari,(p,&1,1))`
`end-if ID`	`end-if ID=1`	`  )`

`then-actions` (or `else-actions`) are executed only when the condition is true (or false). This sequence of 3AK instructions are in line with conditional code representation presented in section 4.

These 3AK instructions are suitable for most of optimization techniques, like constant folding, dead and unreachable code elimination, register allocation and much more. They can be also easily translated into ALLL code as was shown in the example.

7 CONCLUSION

We presented 2 languages and a compiler between these languages. The following work will include increasing performance of the generated programs by implementing some optimization techniques. We already started this part of the project. The goal is to make a compiler able to generate optimized ALLL code as effective (or more) as direct implementation of the algorithm in ALLL.

ACKNOWLEDGEMENT

This work was supported by the European Regional Development Fund (ERDF) within the frame of project Center of excellence IT4Innovations (CZ.1.05/1.1.00/02.0070) as research plan MSM0021630528 and project FIT-S-11-1.

REFERENCES

[1] Frantisek Zboril Jr., Vladimir Janousek, Radek Koci, Frantisek Zboril, and Zdenek Mazal. Framework for model-based design of multi-agent systems. *Int. J. Autonomic Comput.*, 1:140–162, April 2009.

[2] Steven S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[3] Peter Wegner. Concepts and paradigms of object-oriented programming. *SIGPLAN OOPS Mess.*, 1:7–87, August 1990.

[4] František Zbořil. *Plánování a komunikace v multiagentních systémech*. dizertačná práca, Brno, FIT VUT v Brně, 2004.

[5] František Zbořil, Radek Kočí, V. František Zbořil, Vladimír Janoušek, and Zdeněk Mazal. T-mass v.2, state of the art. In *Second UKSIM European Symposium on Computer Modeling and Simulation*, page 6. IEEE Computer Society, 2008.