ISSN 1407-7493 COMPUTER SCIENCE DATORZINĀTNE 2008-7493

INFORMATION TECHNOLOGY AND MANAGEMENT SCIENCE
INFORMĀCIJAS TEHNOLOĢIJA UN VADĪBAS ZINĀTNE

BLACKBOARD ARCHITECTURE PROGRAMMING FOR PRODUCT LIFE CYCLE STAGE DEFINITION

Darya Plinere, Mg.sc.ing., Ph.D. student, Institute of Information Technology, Riga Technical University, 1 Kalku Str., Riga LV-1658, Latvia, e-mail: to.darya@inbox.lv

Keywords: product life cycle, stage definition, blackboard, programming

1. Introduction

Nowadays product life cycle stage definition is very common, but it is not easy to tell which stage the product is in. There are different ways how to define the stage of product life cycle, one of them is to apply the blackboard architecture.

Thanks to its benefits, the blackboard architecture can be successfully used for product life cycle stage definition.

The blackboard architecture can be modeled using different programming languages, for example C++, JAVA or LISP. This paper describes how blackboard architecture was modeled by different authors and offers blackboard architecture programming for product life cycle stage definition using Borland Delphi.

This paper consists of five sections. Section 2 describes blackboard architecture programming, where different authors present their choice of programming language or tool which was successfully used. Section 3 focuses on blackboard architecture programming using Borland Delphi. Section 4 outlines directions for future work, and Section 5 provides conclusions.

2. Blackboard architecture programming

There are different ways how to program the blackboard architecture. Some of examples of what different authors have chosen are listed below.

In [1] a definition of the Reflective Blackboard architectural pattern that is built from the composition of two well-known architectural patterns: the Blackboard pattern and the Reflection pattern, is given.

According to [1], tuplespace architectures are a classical implementation of blackboard architectures. Consider that there are a group of processors that produce pieces of data and a group of processors that use the data. The producers post their data as tuples in the space, and the consumers then retrieve data from the space that match a certain pattern. Tuple space may be thought of as a form of distributed shared memory. [2] TSpaces is a well-known tuplespace architecture that implements the Reflective Blackboard pattern. TSpaces is a Linda-like blackboard architecture for network communication with database capabilities. It provides

group communication services, database services, and event notification services. The TSpaces Event notification engine plays the role of the MOP (Meta-Object Protocol) component of the Reflective Blackboard pattern.

TSpaces reactions are called callback objects and TSpaces meta-data contains information about the operation and the data monitored by the event engine. When implementing MAS (multi-agent systems), the TSpaces event monitoring services are used to establish control strategies. MASs implement this by registering events that notify agents that relevant data has been written on the blackboard. Since the agents are notified of a specific event, the associated control strategy is performed.

MARS (Mobile Agent Reactive Spaces) is another implementation of the proposed pattern. It defines Linda-like blackboards, which can be programmed to react with specific actions to the accesses made by the agents. MARS is implemented using the JavaSpaces technology. MARS was created to help in defining coordination strategies in mobile agent applications. MARS meta-data are called meta-tuples and contain information about the agent that performs a specific operation over specific pieces of data. The MOP protocol is implemented using template-matching searches on a meta-level blackboard where meta-data is stored.

TuCSoN is a coordination model that can be thought of as an implementation of the Reflective Blackboard pattern. This model is based on the notion of tuple centers, which are in fact programmable blackboards. Tuple centers are programmed by associating reactions to specific data and operations. Reactions are created using a proprietary specification language and are handled separately from application basic logic and data [1].

In [3] blackboard system generation in C++ is proposed. According to [3], there are currently several existing systems that auto-generate code for use in blackboard systems (Ho, 1991; Ho, 1994; Hewett, 1993; and McManus, 1992) that create a serial implementation of a blackboard system. These serial implementations do not exploit the parallelism inherent in the blackboard paradigm. These systems were validated using a simulation of a parallel system on a serial processor.

The COBS Blackboard System Code Generator produces code in Common Lisp using the Common Lisp Object System and the Common Lisp Interface Manager and is portable to any computer that supports the Common Lisp Standard.

The Blackboard Generator in C++ (BBG++) is an auto-code generator for a C++ concurrent blackboard system. The generated code includes all necessary source codes for the blackboard system, with the exception of the knowledge source functionality provided by the user [3].

In [4] a definition of the event-based blackboard architecture is proposed which is constructed by the integration of two popular architectural patterns: the Blackboard pattern and the Implicit Invocation pattern.

T Spaces is an application of event-based blackboard architecture. It is a middleware component which provides group communication and event notification services. Its tuple spaces (blackboard) hold data to be exchanged. A set of APIs (application programming interfaces) implements the function of control policies by means of implicit invocation. The data producer and consumer (knowledge sources) do not know each other. Thus, they can be attached to (detached from) the system easily by registering (removing) interests in events.

JavaSpaces is designed to provide a simple unified mechanism for dynamic communication, coordination, and data- and object-sharing. It also implements the event-based blackboard architecture. Data or objects are stored in the Spaces, shared by providers and requesters in the way of implicit invocation [4].

The simple blackboard system (SBB) of [5] is written in Common Lisp. According to [4], it allows KSs (Knowledge sources) to be implemented as Common Lisp functions. KSs

can be triggered in response to the creation of particular blackboard objects (represented as structures). The blackboard is represented as a simple list of objects (a hash table would be a better choice!), and no retrieval capabilities have been provided. The control component supports only one scheduling approach: last-in, first-out [5].

The radar tracking blackboard (RTBB) [6], the subject of this tutorial exposition, is constructed in CLOS (Common Lisp Object System) with KS's written either in Common Lisp or in C [6].

ARBS (an algorithmic and rule-based blackboard system) was originally developed in 1990, it was written in Pop11 and designed to operate under Unix. The implementation in Pop11 compromised performance. For tackling increasingly complicated engineering problems, in [7] the authors decided to re-design a new distributed version of the software, DARBS, implemented in standard C++ [7].

3. Blackboard architecture programming using Borland Delphi

The use of blackboard architecture for product life cycle stage definition was proved in the author's previous paper [8]. The next step was blackboard architecture implementation. It is shown above that the blackboard architecture can be programmed using different programming languages. In this paper, blackboard architecture programming using Borland Delphi programming language is offered.

This is the first release of the blackboard architecture implementation for product life cycle stage definition. The main idea is to program and test available rules in order to know if they work correctly in the program.

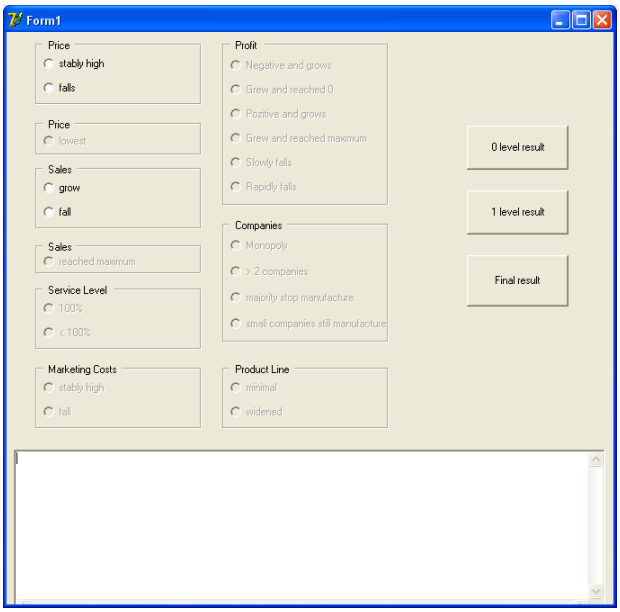

Figure 1. The user interface

The user interface is shown in Figure 1. The user should mark the data he has. First, the user should mark visible information about the price and sales, then another data become visible. This is done because of the high importance of this data. There is a check of input data in the program. The initial data is recorded in the blackboard and can be seen below in the window (clicking on the button "0-level result") in binary form. 0-level means that this is initial data and it has been written on the blackboard.

The 1-level result means that partial solution can be written in the window below. The final result will show the result and rule numbers that have provided this solution (see Figure 2).

The idea how it works is as follows: initial data are recorded on the blackboard. Clicking button “1-level result” knowledge sources are executed and partial result is written on blackboard and in the window below as well. Knowledge sources are looking for the opportunity to make a contribution, they try to find in initial data the data they need for their rules. Final result execution works approximately similarly as 1-level result execution. For final result the intermediate result is needed.

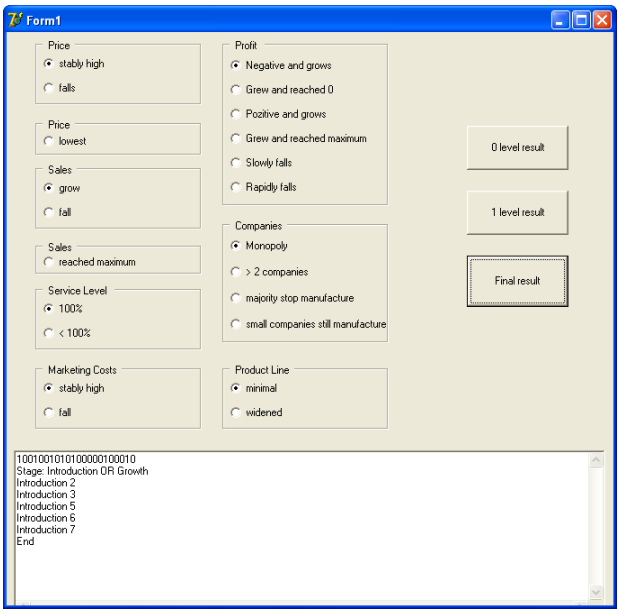

Figure 2. The result of stage definition

If the data marked is incorrect, then the answer will be mutually contradictory (see Figure 3). If the answer cannot be found, the program writes “End” without final solution, which means that the program stopped working without finding the final result.

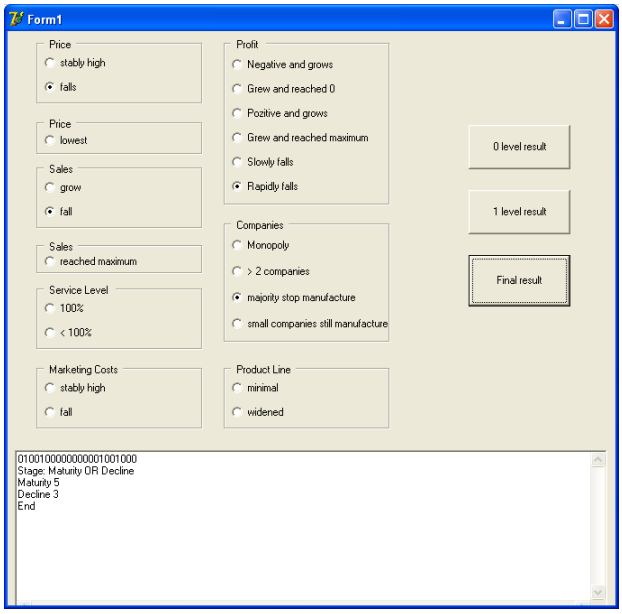

Figure 3. The result of the contradictory data marked

The program works properly, but it should be noted that some rules have been changed during the test. Therefore it can be concluded that the main purpose of this first release is achieved.

When solving this problem, Borland Delphi has demonstrated effectiveness and visibility, and simplicity for user.

4. Future work

The main purpose of this first release was to correctly program working program and to test available rules. This purpose is achieved and the future work is going to be an increase in rule quantity, to make explanation facility, and receiving the input data from another program (charts, statistics, etc.)

5. Conclusions

Product life cycle stage definition is very common. Companies profit depends on which of the stages the product is in. There are different methods for product life cycle stage definition; the proposed one is blackboard architecture application. The blackboard architecture can be used to solve different problems; it is proved that it can be successfully used for product life stage definition.

The blackboard architecture can be programmed using different programming languages and tools. Mostly C++, Java or LISP languages are used for that purpose.

This paper presents successful blackboard architecture programming for product life cycle stage definition using Borland Delphi programming language.

The proposed system has proved its effectiveness and should be studied deeper. The main directions of future work are outlined above.

References

1. Silva O.; Garcia A.; Lucena C. “The Reflective Blackboard Architectural Pattern for Developing Large Scale Multi-Agent Systems” *Proceedings of the First International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*. Pp. 73-93, May 2002.
2. Wikipedia, free encyclopedia, http://en.wikipedia.org/wiki/Tuple_space.
3. Mcmanus J.W.; Kaplan J.A.; Bynum W.L. “Automated Concurrent Blackboard System Generation in C++”, Langley Research Center NASA Technical Document NASA/TM-1999-209128, 1999.
4. Dong, J.; Chen, S.; Jeng, J.-J. “Event-Based Blackboard Architecture for Multi-Agent Systems” Information Technology: Coding and Computing, 2005. ITCC 2005. Volume: 2, pp. 379- 384, ISBN: 0-7695-2315-3.
5. Corkill D.D. “Blackboard systems”, AI Expert 6 (9) (1991) 40-47.
6. Kersten P. R.; Kak A. C. “A Tutorial on Lisp Object-Oriented Programming for Blackboard Computation”. International Journal of Intelligent Systems. Volume 8 Issue 5. Pages 617 – 669. Published Online: 13 Mar 2007.
7. Nolle, L.; Wong, K.C.P.; Hopgood, A.A. “DARBS: A Distributed Blackboard System”. In: 21st SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence, 10-12 December 2001, Cambridge, UK.
8. Rudenko D.; Borisov A. “Blackboard architecture for product life cycle stage definition”, „MENDEL 2008 - 14th International Conference on Soft Computing", edited by Matoušek Radomil. ISBN: 978-80-214-3675-6, pp.252-257, June 18-20, 2008, Brno, Czech Republic.

Plinere Darja. Blackboard arhitektūras programmēšana produkta dzīves cikla fāzes noteikšanai

Produkta dzīves cikla koncepcija - katrs produkts pāriet pāri četrām attīstības fāzēm: ievads, pieaugums, briedums un pagrimums. Tajā pat laikā kad produkts attīstās un pāriet pāri šīm stadijām kompānijas peļņa mainās, dažādas stratēģijas ir jāizmanto, lai garantētu ka produkts ir pieprasīts savā tirgū, tāpēc ir liela nepieciešamība savlaicīgi noteikt kurā no fāzēm pašlaik ir produkts. Blackboard arhitektūra var būt veiksmīgi pielietota produkta dzīves cikla fāzes noteikšanai. Blackboard arhitektūra tika izgudrota kā līdzeklis sarežģīto un slikti noformulēto uzdevumu risināšanai. Pirmā blackboard arhitektūra - runas atpazīšanas sistēma Hearsay ir parādījusies pirms 30 gadiem. Vairākums no blackboard arhitektūrām var būt nomodelēti izmantojot C++, Java vai LISP programmēšanas valodu. Šis raksts prezentē dažādu autoru darbu apskatu ar piedāvātām programmēšanas valodām un sistēmām blackboard arhitektūras realizācijai. Piedāvātās blackboard arhitektūras zināšanas avotos ir likumi produkta dzīves cikla fāzes noteikšanai. Šis raksts prezentē arī ieprogrammētu un notestētu blackboard arhitektūras sistēmu, likumu darbības pārbaudei un kura var būt pielietota produkta dzīves cikla fāzes noteikšanai.

Darya Plinere. Blackboard architecture programming for product life cycle stage definition

The product life cycle concept suggests that a product passes through four stages of evolution: introduction, growth, maturity and decline. As the product evolves and passes through these four stages, profit is affected, and different strategies have to be employed to ensure that the product is a success within its market, therefore there is a need to define in time in which of the stages the product is at the moment. The blackboard architecture can be successfully used for the definition of the stages in product life cycle. The blackboard architecture was originally designed as a methodology aimed to handle complex, ill-defined problems. The first famous example - the Hearsay II speech recognition system appeared about 30 years ago. Most blackboard architectures can be modelled using C++, Java or LISP programming languages. This paper presents an overview of different authors' works with proposed programming languages and systems for blackboard architecture implementation. Knowledge sources of the proposed blackboard architecture consist of rules for product life cycle stage definition. The paper also presents a programmed and tested system of blackboard architecture for checking work of rules that can be used for product life cycle stage definition.

Плинер Дарья. Программирование архитектуры классной доски для определения фазы жизненного цикла товара

Концепция жизненного цикла продукта предполагает, что продукт проходит четыре фазы развития: введение, рост, зрелость и спад. В то время, когда продукт развивается и проходит через эти четыре фазы, меняется прибыль; чтобы гарантировать, что продукт имеет успех в пределах своего рынка, необходимо применять различные стратегии, поэтому существует необходимость вовремя определить, в какой из фаз находится продукт в настоящее время. Архитектура классной доски может успешно использоваться для определения фаз жизненного цикла продукта. Архитектура классной доски была изобретена как средство для решения сложных и плохо сформулированных задач. Первый известный пример – система распознавания речи Hearsay - появилась около 30-ти лет назад. Большинство архитектур классной доски может быть смоделировано, используя языки программирования, такие как C++, Java или LISP. В данной статье представлен обзор работ разных авторов с применением разных языков программирования и систем для реализации архитектуры классной доски. Источники знаний предложенной архитектуры классной доски содержат правила для определения фазы жизненного цикла товара. В статье также представлена запрограммированная и протестированная система архитектуры классной доски для проверки работы правил, которая может использоваться для определения фазы жизненного цикла продукта.