XML 기반 강건 타입형 유전자 프로그램의 이식·독립적 표현

(XML-based Portable Self-containing Representation of Strongly-typed Genetic Program)

이 승 익 † Ivan Tanev†† Katsunori Shimohara†††
(Seung-Ik Lee) (Ivan Tanev) (Katsunori Shimohara)

요 약 선택과 재생산을 특징으로 하는 계통적 학습에서 유전자 프로그램이 가지는 긴 설계시간/높은 계산노력/낮은 계산효율을 극복하고자, 이 논문은 XML에 기반을 둔 유전적 표현 방법을 제안한다. 이 방법에서 유전자 프로그램과 유전자 연산은 기성 DOM 파서의 API 호출에 의하여 관리되기 때문에, 유전자 프로그램을 설계하는데 소비되는 시간이 상당히 단축되는 특징이 있다. 또 표준 XML 스키마를 기반으로 의미적으로 올바른 유전자 프로그램만을 다루기 때문에 탐색공간과 계산노력이 감소된다. 그리고 이형 분산 컴퓨팅 환경에서 유전자 프로그램의 이주에 적합한 시스템 및 형식인 XML을 사용하기 때문에 유전자 프로그램이 병렬적으로 수행될 수 있고, 이에 따라 계산효율이 향상된다. 제안된 방법의 검증을 위하여 포식자-피식자 문제에서 다중 에이전트의 사회적 행동의 진화에 적용한 결과, 유전자 프로그램에 대한 계산시간이 단축됨을 보인다.

키워드 : 유전자 프로그램, XML, DOM, 분산 컴퓨팅, 에이전트

Abstract To overcome the long design time/high computational effort/low computational performance of phylogenetic learning featuring selection and reproduction, this paper proposes a genetic representation based on XML. Since genetic programs (GP) and genetic operations of this representation are maintained by the invocation of the built-in off-the-shelf XML parser's API, the proposed approach features significant reduced time consumption of GP design process. Handling only semantically correct GPs with standard XML schema can reduce search space and computational effort. Furthermore, computational performance can be improved by the parallelism of GP caused by the utilization of XML, which is a feasible system and wire format for migration of genetic programs in heterogeneous distributed computer environments. To verify the proposed approach, it is applied to the evolution of social behaviors of multiple agents modeling the predator-prey pursuit problem. The results show that the approach can be applied for fast development and time efficiency of GPs.

Key words : Genetic program, XML, DOM, Distributed computing, Agent

1. 서 론

유전자 프로그램(Genetic Programming)[1]은 적합 개체의 생존과 재생산이라는 종의 자연 진화 메커니즘에서 영감을 받아 이루어진 영역 독립적인 문제 해결 알고리즘이다. 특정한 영역에 구애받지 않아서, 아날로그/디지털 회로 디자인, 시공간적 정보 식별과 예측, 그리고 기계학습 등과 같은 인공지능의 여러 분야에서 어려운 문제를 해결할 수 있는 방안을 제공하였고, 최근에는 다중 에이전트 시스템과 관련하여 각광받고 있다. 이의 원인으로는 연역적 접근 방식으로는 창발적 행동(결과)을 유도하기가 부적절하다는 점과[2] 사람이 설계하는 것보다 최적화될 수 있다는 점을 들 수 있다[3].

다중 에이전트 시스템은 다수의 에이전트가 상호작용을 통하여 시뮬레이션, 집단적 행동, 또는 합성적 세계

· This research was conducted while the 1st author worked in ATR Human Information Science Labs and supported in part by the National Institute of Information and Communications Technology.
† 정 회 원 : 한국전자통신연구원
the_silee@hanmail.net
†† 비 회 원 : Faculty of Engineering, Doshisha University/ Network Informatics Labs., ATR International
itanev@mail.doshisha.ac.jp/i_tanev@atr.jp
††† 비 회 원 : Network Information Lab., ATR International
katsu@atr.jp
논문접수 : 2003년 8월 12일
심사완료 : 2005년 2월 24일

를 구성하는 시스템이다. 정보화로 인하여 사회구성원끼리의 상호작용이 폭발적으로 증가하는 시점에서, 이러한 상호작용을 모델링, 예측, 분석할 수 있는 다중 에이전트 시스템은 점점 중요해지고 있으며, 응용분야도 분산 인공지능, 웹 컴퓨팅, 웹 기반 전자 상거래, 로보틱스 및 인공생명에 이른다[4].

다중 에이전트 시스템과 유전자 프로그램의 결합은 다중 에이전트 시스템의 행동을 시뮬레이션을 통하여 계통발생 시킨 후 이를 다시 시뮬레이션이나 실제 하드웨어에 탑재하여(몇몇 진화 로보틱스의 경우), 잡음/불확실/동적 변화하는 환경에 대한 개체 발생적 또는 후생적(epigenesis) 적응을 목적으로 한다. 하지만 이것이 성공하기 위해서는 유전자 표현, 유전자 연산, 평가함수와 같은 유전자 프로그램의 속성이 올바르게 설정되어야 하고 유전자 프로그램의 긴 개발기간 및 계산시간에 대한 적절한 대처가 요구되고 있다.

개발기간이 장기화되는 주된 원인은 다음과 같다. 우선, 후보 해 표현, 초기 집단, 유전자 연산, 적합도 함수와 같은 유전자 프로그램의 주요 속성에 대하여 3세대 프로그램 언어들의 포괄적인 지원이 부족하고 관련 소프트웨어 표준이 없는 점이다. 또, 공개되어 있는 유전자 프로그램 시스템 중에는 요구사항에 맞도록 변경 가능한 시스템이 있기는 하지만[5,6], 적합도 평가나 에이전트와 환경과의 상호작용이 영역 의존적인 점, 그리고 적합도 평가의 부수효과 때문에 공개 시스템의 실제적인 사용은 아직 문제점을 안고 있다. 계산시간이 길어지는 점은 다양한 샘플링 환경에서(수십에서 수백), 많은 세대(수백에서 수천)동안, 많은 수의 후보 해를(수백에서 수천) 유전자 프로그램이 진화해야 함에 기인한다. 이는 실세계의 온라인 진화대신에 계통발생적인 오프라인(시뮬레이션) 진화가 가지는 속도의 장점을 감소시킨다.

이 논문은 유전자 프로그램의 활용의 걸림돌이 되고 있는 개발기간 및 계산시간의 장기화 문제를 해결하고자 한다. 첫째, 개발기간을 단축하기 위하여 산업 표준인 문서객체 모델 (Document Object Model, DOM)과 확장 마크업 언어(eXtensible Markup Language, XML), 그리고 이를 지원하는 기성 소프트웨어를 활용하는 유전적 표현방법이 제안된다. 둘째, 계산시간을 단축하기 위해서 이 논문은 계산노력과 계산효율이라는 관점에서 접근한다. 유전자 프로그램에 강건형 타입을 [7,8] 결합하여 탐색공간을 축소함으로써 정해진 성능에 이르기까지 평가되는 유전자 프로그램의 수를 감소시켜 계산노력을 절감한다. 또, 유전자 프로그램을 병렬적으로 실행하여 단위시간에 평가되는 개체의 수, 즉 계산효율을 향상한다. 마지막으로, 이 논문은 제안된 방법을 다중 에이전트 시스템에 적용하여 그 유효성을 검증하고 미래의 유전자 프로그램 시스템의 핵심적인 속성이 될 수 있는 가능성 알아본다.

이 논문의 구성은 다음과 같다. 2장은 적응적 다중 에이전트 시스템상의 오프라인 계통발생과 개체 발생적 학습에서 알고리즘적 패러다임인 유전자 프로그램에 대하여 소개한다. 또, 유전자 프로그램을 XML로써 표현하는 방법과, 개발기간과 계산시간을 단축하기 위한 방법에 대하여 설명한다. 3장은 제안한 방법을 검증하기 위해서, 포식자-피식자 문제를 다룬 다중 에이전트 시스템에서의 에이전트의 사회적 행동을 진화한다. 4장은 이 논문의 결론에 대하여 다룬다.

2. 제안하는 방법

이 장은 XML을 이용하여 유전자 프로그램을 표현하고 이를 산업 표준 및 기성 소프트웨어를 이용하여 변경/관리함으로써 유전자 프로그램의 개발기간을 단축하는 방법에 관하여 논한다. 또한 강건 타입형 병렬 분산 유전자 프로그램 시스템을 활용하여 계산시간을 단축하는 방안을 제안한다.

2.1 알고리즘적 패러다임

다중 에이전트 시스템에서 자율성과 적응성을 지닌 에이전트의 반응형 행동을 제어하기 위해서는 자극-반응 형태의 규칙집합이 자연스럽다[9]. 이를 사람에 비유하면, "비가 오면 우산을 쓴다."가 자극-반응 규칙의 하나의 예가 된다. 여기서 "비가 오면"은 에이전트를 둘러싼 환경에서 인지되는 자극이 되며, "우산을 쓴다"는 "비가 오면"의 자극에 대응하는 에이전트의 반응이다. 일반적으로 유전자 프로그램이 적용될 수 있는 대상 중의 하나가 자극에 대한 최적의 반응을 결정하기가 어렵다는 점일 것이다. 다시 말하면, 비가 오면 우산을 쓰는 것이 가장 좋은지 아니면 우비를 입는 것이 좋은지 결정하기가 쉽지 않다는 것이다. 따라서 이 논문은 전역적 탐색 능력이 있는 유전자 프로그램을 이용하여 자극과 반응의 관계를 최적화한다.

유전자 프로그램에서 하나의 개체는 파스트리로 표현된다. 파스트리의 노드는 함수, 변수, 또는 상수이다. 함수는 부트리를 매개값으로 갖는 비단말 노드이다. 함수는 산술 연산자와 논리 연산자는 물론 IF-THEN과 같은 조건절을 표현할 수 있기 때문에, 자극과 그에 상응하는 반응과의 관계설정에 용이하다. 변수와 상수는 단말 노드로 표현된다.

그림 1은 자극-반응 규칙의 예를 보인다. 그림 1(a)는 에이전트 Peer_d로부터 거리가 20(예를 들면, 20미터)미만이면, 에이전트 Peer_a의 방향에다 10(예를 들면, 10도)을 더한 방향으로의 전환을 표현하는 의사코드를 보인다. 그림 1(b)는 1(a)에 상응하는 파스트리를 보인다. 그림 1(c)는 파스트리를 사람이 읽기 쉬운 S-표현으로 표현한다.[1)]

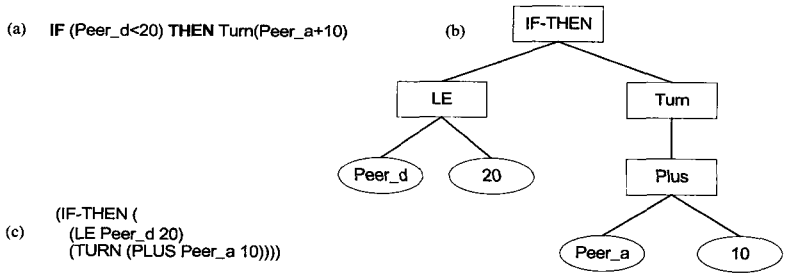


그림 1 에이전트의 행동을 제어하는 규칙의 예: (a) 의사코드 표현 (b) 파스트리 (c) S-표현

전형적으로, 유전자 프로그램의 파스트리 또는 그에 상응하는 동등표현(S-표현, 전위 또는 후위 표현)은 유전자 프로그램에 의하여 고유의 방식으로 유지/조작되므로, 일반적 목적의 유전자 프로그램이 특정 영역에(예를 들면, 다중 에이전트 시스템) 적용되기 위해서는 영역 의존적인 부분에 대한 수정이 필요하다. 하지만, 산업 표준과 기술이 유전자 프로그램의 주요한 속성을 지원한다면 시간 소비적인 수정이 방지되거나 감소하므로 이를 바탕으로 유전자 프로그램을 설계/구축하는 것이 보다 빠르고 유연할 것으로 보인다.

2.2 XML을 이용한 유전자 프로그램의 표현

이 절은 산업 표준과 기술을 이용하여 유전자 프로그램을 표현하는 방법에 대하여 다룬다. 우선적으로 유전자 프로그램의 표현에 접근하는 이유는, 그것이 모든 진화 알고리즘의 기본적인 속성이고 해당 시스템의 소프트웨어 공학 과정과 성능에 영향을 미치기 때문이다.

이러한 점에서 최근에 광범위하게 인정받고 있는 DOM/XML[10]의 유연성에 착안하여, 텍스트 기반 XML과 메모리 내부의 DOM 트리로 유전자 프로그램을 표현하는 방법을 제안한다. DOM/XML 기술은 컴퓨터 구조[11], 소스 코드[12], 에이전트 통신 언어[13], 그리고 시맨틱 웹(Semantic Web)[10] 등의 표현에 광범위하게 확산되고 있음에도 불구하고, 아직까지 유전자 프로그램 같은 진화구조의 표현에는 적용되지 않고 있다. 이 논문에서 유전자 프로그램은 메모리 내부에서는 DOM 파스트리로 표현되며, 유전자 연산 또한 언어 중립적인 DOM 파서를 사용하여 DOM 파스트리에서 수행된다. 유전자 프로그램의 또 다른 표현인 XML은 분산 유전자 프로그램 환경에서 컴퓨팅 노드사이의 이주를 위하여 이용된다.

DOM 파스트리는 일반적인 트리 기반 유전자 프로그램의 파스트리에 정식으로 상응하므로 유전자 프로그램을 표현하기에 적합하다. 일반 파스트리의 비단말 노드(예를 들면, IF-THEN 노드)는 DOM 트리에서 nodeType 속성이 NODE_ELEMENT인 비단말 노드로 표현되고, nodeName 속성은 일반 파스트리에서 비단말 노드의 텍스트(예를 들면, IF-THEN)에 상응한다. 일반 파스트리의 단말 노드는 DOM 트리에서 nodeType이 NODE_TEXT이고, nodeName이 일반 파스트리에서 상응하는 단말의 의미(예를 들면, PERCEPTION, CONSTANT 등)를 표현하고, nodeValue는 상응하는 단말 노드의 심벌(예를 들면, Peer_d, 20, Peer_a, 10)을 가진다. 예를 들면, 그림 1에 보인 규칙에 상응하는 DOM 트리와 XML은 그림 2(a)와 그림 2(b)로 각각 표현된다.

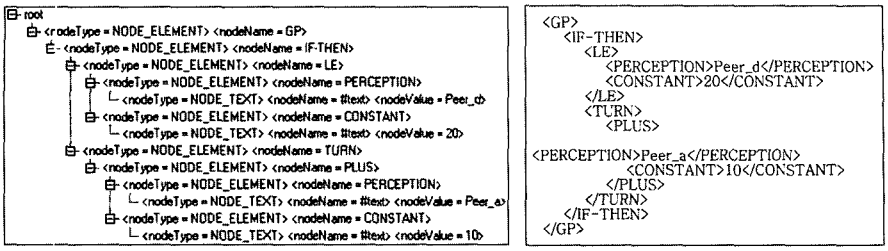
2.3 개발기간의 단축

고유의 코드와 표현방법을 이용하여 파스트리를 조작하는 전통적인 접근방식과는 대조적으로, XML 기반 접근방식은 유전자 프로그램을 개발하는데 소용되는 시간을 최대한 단축시킬 수 있다. 이는 첫째, 소프트웨어 플랫폼, 개발 언어, 그리고 프로그래밍 패러다임을 개발자가 임으로 선택할 수 있으므로 별도로 개발자가 습득해야할 필요성이 없다는 점과 둘째로, 기성 소프트웨어 툴을 사용하므로 새롭게 개발해야 하는 부분이 감소한다는 점에 기인한다(표 1 참조).

여기에서 개발기간 단축의 가장 큰 원인은 코드 재사용에 의한 것이다. 프로그래밍 언어가 어셈블리 언어→절차적 언어→모듈 언어→객체지향 언어→컴포넌트 객체 언어로의 과정을 거쳐 오는 동안 코드 재사용에 의하여 개발기간의 단축을 이루어 왔음은 주지의 사실이다. 이 논문에서도 이러한 점에 착안하여 컴포넌트 객체 모델을 유전자 프로그램의 표현(XML을 이용) 및 유전자 연산(DOM 파서 이용)에 도입함으로써 기존의 컴포넌트 객체를 재사용할 수 있게 되어, 개발기간이 단축될 수 있다.[2]

1) 여기서 S-표현은 전위 폴리쉬(prefix polish) 표현의 변형으로서, 파스트리의 함수노드는 *"(함수이름 매개값로 쓰이는 부트리의 S-표현)"*로 표현된다.

2) 이 주장을 뒷받침할 수 있는 정략적인 자료의 제시가 요구되어 지나, 문제의 특성상 정량적인 분석 데이타의 확보가 난해하고, 코드의 재사용에 의한 개발기간의 단축은 일반적인 프로그래밍 언어의 발전과정의 예에서 명확히 유추될 수 있으므로 이에 대한 정략적인 자료의 제시는 생략한다.



(a) DOM 파스트리 (b) XML 표현

그림 2 유전자 프로그램의 DOM과 XML 표현

표 1 유전자 프로그램의 개발기간 단축 요인

단축 요인	설명
DOM 파서의 API 사용	유전자 프로그램의 파스트리는 DOM 파서 고유의 표준적인 API에 의하여 조작되므로 별도의 API를 개발/테스트/디버깅할 필요가 없음
플랫폼 중립적인 파서	DOM 파서는 대부분의 소프트웨어 플랫폼에서 이용가능하기 때문에, 서로 다른 플랫폼(예를 들면, 자바 클래스)사이의 유전자 프로그램 코드 및 데이타 구조의 이식이 용이
언어 중립적 파서	DOM 파서는 언어 중립적인 부품(예를 들면, 마이크로소프트 ActiveX)으로서 이용가능하기 때문에, 유전자 프로그램에 사용된 언어에 관계없이 같은 프로그래밍 모델을 사용하여 파스트리에 대한 조작가능
패러다임 중립적 파서	DOM 파서는 독립실행 응용 프로그램, 웹 클라이언트(예를 들면, 웹 브라우저), 웹 서버, 그리고 데이타베이스 서버와 같은 다양한 응용 소프트웨어 패러다임에서 재사용 가능한 부품으로서 이용가능
다양한 소프트웨어 툴	XML 저작을 위한 대부분의 툴은 좋은 사용자 인터페이스와 시각화, 직관적이고 이해하기 쉬운 방식의 직접적 조작방법을 제공

2.4 계산시간의 단축

개발기간의 문제와 더불어 유전자 프로그램의 또 다른 문제점으로 지적되는 사항은 바로 계산시간이 오래 걸린다는 점이다. 이 절에서는 이러한 문제점을 해결하기 위하여 계산노력과 계산효율이라는 두 관점에서 접근한다. 계산노력은 정해진 수준의 성능을 얻기까지 평가되는 총 개체의 수이고, 계산효율은 단위시간에 평가되는 개체의 수로 정의된다.

2.4.1 계산노력의 감소

이 절에서는 계산노력의 감소를 위하여, XML 방식에 기반을 둔 강건 타입형 유전자 프로그램[7]을 구현하는 방법에 대하여 자세히 언급한다. 별다른 제약 없이 임의의 복잡성을 가진 자극-반응의 규칙 집합은 탐색공간이 광대하고 다차원적이어서 많은 계산노력을 필요로 한다. 특히, 임의적인 부트리의 생성이나 수정은 (a) 연산 및 함수에서 데이타 형의 불일치,[3] (b) 센서의 감지 범위를 초과한 값의 사용[4] 등 의미론적으로 무의미한 부트리가 나올 수 있다.

이 논문은 사전지식과 합당하지 않은(다시 말하면, 의미론적으로 난센스인) 부분이 발생하지 않도록 유전자 프로그램의 문법 구조에 제약을 가함으로써 탐색공간에 대한 가지치기를 수행한다[2]. 이는 변수와 상수에 대한 데이타 타입, 그들이 가질 수 있는 값의 범위, 연산결과 값에 대한 데이타 타입 정의, 그리고 피연산자에 대한 데이타 타입의 정의 등을 속성으로 지니는 강건타입형 유전자 프로그램을 통해서 이루어진다. 이를 위한 필요조건과 구현방법은 표 2와 같다.

World Wide Web 컨소시엄(W3C)의 권고 표준안인 XML 스키마[14,15]는 문서에서 나타날 수 있는 항목을 정의하고 이것이 유효하기 위해 준수해야 할 구조를 규정한다. 그러므로 XML 스키마는 유전자 프로그램을 위한 항목의 정의와 강건 타입의 속성을 갖는 유전자 프로그램을 위한 구조를 정의하기에 적당하다.

유전자 프로그램의 검증은 스키마에 의해 요구되는 문법에 문서가 충실히 따르고 있는가를 확인하는 것이다. 스키마의 문법 또한 XML 표준에 따르고 있기 때문에 유전자 프로그램의 파스트리를 생성하고 변경하는 루틴을 재사용하여 스키마에 접근하고 그 안에 정의된 규칙에 따라 유전자 프로그램에 규칙을 강제할 수 있다.

3) 예를 들면, 거리 값과 각도 값의 불리언 표현은 무의미하다. 마찬가지로, Turn() 함수의 의미상 angle형 이외의 데이타 형을 가진 매개값은 의미론적으로 무의미하다.

4) 예를 들면, 감지범위가 400이하인 센서와 1000이상의 센서 값과의 부울 표현. 에이전트의 인식은 시뮬레이션 되고 있는 센서의 물리적 능력에 제약이 있기 때문이. 주어진 범위 내의 인식 값 이외에는 의미가 없다.

표 2 강건 타입형 유전자 프로그램을 위한 필요조건과 구현방법

필요조건	구현 방법
강건 타입형 유전자 프로그램의 문법 표현	XML 스키마
문법적으로 올바른 유전자 프로그램의 생성/변경	XML 스키마와 DOM 파서의 API
문법과의 일치성 검증	DOM 파서의 자동검증 기능

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="GP">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="IF-THEN" type="IF-THEN"/>
    ...
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="IF-THEN">
 <xs:sequence>
   <xs:element name="COND-THEN" type="COND-THEN"/>
    ...
</xs:sequence>
</xs:complexType>
<xs:complexType name="COND-THEN">
  <xs:choice>
   <xs:element name="COND_TDist" type="COND_TDist"/>
   <xs:element name="THEN"       type="THEN"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="COND_TDist">
<xs:sequence>
 <xs:element name="VAR_TDist"   type="VAR_TDist"/>
 <xs:element name="OPER_TDist"  type="OPER_TDist"/>
 <xs:element name="CONST_TDist" type="CONST_TDist"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="VAR_TDist">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Wall_d"/>
  <xs:enumeration value="Peer_d"/>
   ...
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OPER_TDist">
 <xs:restriction base="xs:string">
```

그림 3 그림 2에서 보인 규칙의 자극 부분에 상응하는 문법을 정의한 XML 스키마의 일부

더욱이, 대부분의 DOM 파서는 특정한 스키마에 대하여 파스트리의 자동적인 검증을 수행할 수 있다.

우리는 표준 Backus Naur Form(BNF) 표현으로 쓰인 유전자 프로그램의 문법을 상응하는 XML 스키마로 자동적으로 변환하는 규칙집합을 개발하였다. 이러한 규칙은 간단하기 때문에, 그림 2에서 보인 규칙의 자극 부분을 규정하는 XML 스키마의 일부를 그림 3에 보이는 것으로 설명을 대신한다.

그림 3은 에이전트의 감지능력을 표현하기 위하여 인식정보의 종류(예를 들면, 벽과 다른 에이전트에의 거리, Wall_d, Peer_d)와 그에 상응하는 센서의 범위(0..400)를 표현한다. 그림 4는 그림 3에서 보인 XML 스키마를 적용하여 생성된 강건 타입형 유전자 프로그램의 XML 표현을 보인다.

유전자 프로그램에 문법을 도입하여 탐색공간을 축소하는 과정에서 영역 의존적인 지식은 에이전트에 대한 형태를 정의하는 부분까지만 사용되었다는 점에 주목하자. 유전자 프로그램의 문법에 부여된 제약점은 포식자 에이전트가 자신의 형태[5]를 완전히 인지하고 있다는 자연스런 가정과 덧셈, 뺄셈, 그리고 비교 연산에서 쓰이는 피연산자는 같은 데이타 형을 가져야 한다는 잘 알려진 규칙에 기반하고 있다. 이러한 제약점은 영역 혹은 에이전트가 존재하는 시뮬레이션 세계에 특정한 선험적 지식

5) 에이전트의 센서와 모터의 물리적 성질. 예를 들면, 센서의 감지거리나 모터의 속도.

```
<GP>
<IF-THEN>
<COND-THEN>
<COND_TDist>
<VAR_TDist>Peer_d</VAR_TDist>
<OPER_TDist>LE</OPER_TDist>
<CONST_TDist>20</CONST_TDist>
</COND_TDist>
</COND-THEN>
<THEN>
...
</THEN>
```

그림 4 그림 3에서 보인 XML 스키마에 상응하는 강건 타입형 유전자 프로그램의 XML 표현

을 포함하지는 않는다. 따라서 제안된 방법론이 강건 타입형 유전자 프로그램이 가지고 있는 패러다임 자체의 영역 독립성을 손상시키는 접근방법이라고 볼 수 없다.

2.4.2 계산효율의 향상

이 절은 계산효율의 향상을 위해서 유전자 프로그램을 병렬 분산적으로 구현하는 방법에 대하여 기술한다. 비록 전용의 소프트웨어 및 하드웨어가 없다고 하더라도 기존의 소프트웨어 및 하드웨어 컴포넌트에 기반을 두어 병렬컴퓨팅 환경을 구축할 수 있기 때문에, 병렬적 구현은 유전자 프로그램의 계산시간의 문제를 해결하는데 가장 바람직한 방법으로 대두되고 있다.

유전자 프로그램의 병렬 분산적 구현은 컴퓨팅 노드 사이의 유전자 프로그램의 이주를 의미한다. 보통의 파스트리 형태로는 이를 관리하는 프로세스의 가상 주소 공간을 넘어서는 아무런 의미가 없기 때문에, 유전자 프로그램이 이주 가능하고 플랫폼 독립적이며 상호 운용성을 갖추어야 한다. 이런 관점에서, 우리는 XML 형식의 텍스트로서 유전자 프로그램 및 스키마를 표현하는 방법을 제안한다. XLM은 통신상에서 플랫폼 독립적인 데이타 표현과 상호 운용성을 위한 시스템 및 형식으로서 필수적인 요구사항이 되고 있다는 점도 우리의 선택을 뒷받침한다[16].

유전자 프로그램이 컴퓨팅 노드 사이에서 이주할 때, 송신노드는 DOM 파서 고유의 특성을 사용하여 DOM 파스트리를 그에 상응하는 XML 형식의 텍스트로 변환하고 이를 수신 노드에 보낸다. 수신노드는 DOM 파서의 고유의 기능을 이용하여 이를 다시 DOM 기반 파스트리로 변환하고 상응하는 스키마에 따르고 있는지를 검증한다. 스키마는 그것이 표현하는 유전자 프로그램들에 비하여 불변하기 때문에, 병렬 분산 강건 타입형 유전자 프로그램 시스템의 초기화 단계에서 모든 노드에 단 한 번의 전송으로 충분하다.

3. 검증

제안된 방법이 유전자 프로그램을 효과적으로 표현할 수 있고, 계산노력의 감소 및 계산효율의 향상에 기여한다는 점을 보이기 위하여, 계통적인 학습을 위한 강건 타입형 유전자 프로그램 의 원형을 구현하고 이를 포식자-피식자[17] 문제를 다룬 다중 에이전트 시스템에 적용하였다. 포식자-피식자 문제를 선택한 이유는, 에이전트의 계속적인 인식과 행동으로 인하여 높은 계산노력과 낮은 계산효율을 보이는 광활한 해 공간을 지니고 있기 때문이다.

3.1 포식자-피식자 문제

포식자-피식자 문제는 명확히 정의되고 널리 알려졌으나 쉽게 해결되지는 않는 특성이 있다. 이 논문에서 포식자와 피식자가 놓이는 세계는 $1600mm \times 1000mm$ 의 원환 형태의 시뮬레이션 세계이다. 네 개의 포식자는 피식자를 사방에서 둘러싸서 포획하는 것이 목표이다. 포식자는 좌우로 원하는 만큼 방향 전환을 할 수 있으며, 속도는 0, 0.25, 0.5, 0.75, 그리고 최고 1.0까지 낼 수 있다. 포식자는 일정한 거리 안에 있는 피식자와 자신에게 가장 가까운 포식자를 감지할 수 있는 근접감지 센서가 있다. 피식자는 포식자가 감지되지 않을 경우에는 방황하도록 프로그램 되었으며, 포식자를 감지한 경우에는 설계자에 의해 부여된 최선의 전략을 사용하여 도주한다. 피식자의 최고속도는 포식자의 최고속도보다 높다. 포식자가 피식자에 접근할 때, 피식자가 눈치 채지 못하도록 하고 또 협동적인 작업을 할 수 있도록, 포식자는 피식자의 근접감지 능력보다 넓은 범위를 감지할 수 있다. 감지능력은 포식자가 좋지만 도주시의 속력은 피식자가 빠르기 때문에 포식자끼리의 상호협력이 없이는 피식자를 잡기 힘들다. 포식자의 진화를 위한 유전자 프로그램의 문법이 XML 스키마 형태로 부록에 수록되어 있다.

3.2 실험 조건 및 결과

3.2.1 실험 조건

강건 타입형 유전자 프로그램에서 쓰이는 함수와 단말 집합이 표 3에 나타나있다. 함수 집합은 IF-THEN 문장, 산술연산자, 그리고 비교연산자로 구성되고, 단말

표 3 강건 타입형 유전자 프로그램의 함수집합과 단말집합

분류		표시	참고
함수집합		IF THEN IF THEN-NA LE, GE, WI, EQ, NE +, -	IF THEN 문장 IF-THEN 문장(예외처리포함) <=, >=, Within, =, <> 산술연산자
단말집합	감지능력	Prey_d, Peer_d Prey_a, Peer_a PreyVisible, PeerVisible	피식자나 가장 가까운 동료와의 거리(mm) 피식자나 가장 가까운 동료와의 각도(degree) 피식자나 가장 가까운 동료가 보일 경우 참
	상태변수	Speed	에이전트의 속도(mm/s)
	일회성 상수	Integer	
	이동능력	Turn() Stop, Go_1.0 Go_0.25,Go_0.5,Go_0.75	방향전환(매개값이 0이상이면 시계방향) 속력을 0으로 함, 속력을 최대로 함 속력을 25%, 50%, 75%로 함

집합은 감지능력과 이동능력 등을 표현한다.

유전자 연산은 선택연산, 교차연산, 부트리 돌연변이, 전위연산이 적용된다. 선택연산은 이진 토너먼트가 사용된다. 교차연산은 강건 타입형 유전자 프로그램의 특성이 유지될 수 있도록 부모개체에서 같은 형의(즉, 같은 태그를 가진) 노드(그리고 부트리)사이에서 적용된다. 부트리 돌연변이도 돌연변이를 위해 선택된 노드가 문법적으로 올바른 부트리와 교체되도록 하여 강건 타입이 유지되도록 한다. 변경하고자 하는 노드의 형을 참조하여 문법으로부터 적용 가능한 규칙 중 하나를 임의로 선택한 후 적용한다. 전위연산(transportation)은 하나의 유전자 프로그램 내에서 같은 데이타 형의 노드 두 개를 선택한 후 이들의 위치를 바꾼다.

유전자 프로그램은 10개의 서로 다른 환경에서 테스트된다. 하나의 환경에서 에이전트의 기본 신진 대사량과 이동시에 필요한 에너지를 포함하는 에너지 손실량, 테스트 종결시점의 피식자와의 거리, 그리고 종결시점까지의 경과 시간을 각각 평균하면 에너지-거리-시간을 축으로 하는 공간상의 반지름 벡터가 결정되는데, 이를 해당 프로그램의 적합도로 한다. 따라서 적합도는 작을수록 좋은 성능을 의미한다. 하나의 환경에 대한 테스트 시간은 물리세계의 300초 이내 또는 피식자가 잡혔을 때까지로 제한된다. 샘플링 시간이 0.5초인 경우 600회의 시간동안 하나의 환경에서 테스트가 수행된다.

시뮬레이션의 종결조건은 다음과 같다. 첫째, 최우수 유전자 프로그램의 적합도가 300[6]이하이거나 주어진 10번의 테스트 환경에서 모두 피식자를 잡았을 경우. 둘째, 최종세대인 100세대에 이르렀을 경우. 셋째, 적합도의 개선 없이 16세대 이상 시뮬레이션이 지속된 경우에 시뮬레이션이 종결된다.

한편, 에이전트는 방황/탐색, 맹목적 추격, 그리고 사회적 행동으로서 둘러싸기의 행동 모드가 포섭구조[18]에 의해 제어된다. 이 논문은 방황/탐색과 맹목적 추격을 에이전트의 기본적인 동작이라고 가정하고(즉, 설계자에 의해 미리 에이전트에 부여됨), 가장 높은 우선순위를 갖는 둘러싸기 행동의 진화에 대하여 집중하고자 한다. 이러한 접근방식은 (a) 피식자가 보일 때 에이전트가 맹목적 추격을 할 것인가 아니면 사회적 행동을 할 것인가의 딜레마, (b) 피식자가 보이지 않을 때 에이전트가 방황/탐색을 할 것인가 아니면 사회적 행동을 할 것인가의 탐험-탐색 딜레마를 해결할 수 있는 능력에 대한 동시적 진화를 가능하게 한다.

제안된 방법의 검증을 위해 윈도우 2000 시스템과 MSXML4.0[7]하에서 실행되는 강건 타입형 유전자 프로그램의 원형이 개발되었다. 파서는 XML 스키마와 관련한 W3C의 최근의 권고안을 지원한다. 원형은 통합 개발 환경인 볼란드 델파이 6.0에서 개발되었고, 이형 클러스터 상에서 보스-작업자 모델[19]을 병렬적으로 구현하였다.

3.2.2 실험 결과

그림 5는 최우수 유전자 프로그램의 예를 XML 형태로 보인다. 여기서 한 가지 주목한 점은IF-THEN 규칙과 더불어 IF-THEN-NA(IF-THEN-"not available") 형태의 예외처리 능력이 있는 규칙이 포함되어 있다는 점이다. 예외는 일종의 사건으로서 IF-THEN 규칙의 조건부에서 불리언 표현이 평가될 때 발생된다. 예를 들면 가장 가까운 동료 에이전트나 피식자가 보이지 않는 상황에서 관련 인식 변수를 포함하고 있는 불리언 표현이 평가될 때 발생한다.[8] 그림 6은 같은 프로그램을 읽

6) 적합도 300이 의미하는 바는, 대략적으로 주어진 시간(600 단위시간)의 절반정도에서 주어진 모든 초기 환경에서 피식자를 잡았을 경우의 적합도에 일치하는 수치이다.

7) 마이크로소프트(Microsoft)에 의해 개발된 DOM 파서로서 플랫폼, 언어, 응용 소프트웨어 패러다임에 중립적인 ActiveX 기술에 기반을 두어 개발되었다.

8) 이러한 상황은 포식자의 시뮬레이션 된 센서의 감지 범위에 제약이 있기 때문에 발생한다.

```
<?xml version="1.0" ?>
<GP xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="..₩SharedResources₩
GPSchema.xsd">
<STM ind="3">
 <IF-THEN-NA ind="4"><COND-THEN-NA ind="5">
  <COND_TVisAngle ind="6">
  <VAR_TVisAngle>Prey_a</VAR_TVisAngle>
  <OPER_TVisAngle>GE</OPER_TVisAngle>
  <CONST_TVisAngle>-19</CONST_TVisAngle>
  </COND_TVisAngle></COND-THEN-NA>
 <THEN ind="13"><STML ind="14"><STM ind="15">
  <IF-THEN-NA ind="16"><COND-THEN-NA ind="17">
   <COND_TDistance ind="18">
   <VAR_TDistance>Prey_d</VAR_TDistance>
   <OPER_TDistance>LE</OPER_TDistance>
   <CONST_TDistance>382</CONST_TDistance>
   </COND_TDistance></COND-THEN-NA>
  <THEN ind="25"><STML ind="26"><STM ind="27">
   <COM-TURN ind="28">
    <COM-TURN-OPERAND ind="29">
    <SIGN>+</SIGN>
    <ANGLE-OP ind="32">
     <VAR_TVisAngle>Prey_a</VAR_TVisAngle>
    </ANGLE-OP></COM-TURN-OPERAND>
    <COM-TURN-OPERAND ind="35">
     <SIGN>-</SIGN>
     <ANGLE-OP ind="38">
      <CONST_TVisAngle_Pos>18</CONST_TVisAngle_Pos>
    </ANGLE-OP></COM-TURN-OPERAND>
    <COM-TURN-OPERAND ind="41">
     <SIGN>+</SIGN>
     <ANGLE-OP ind="44">
      <VAR_TVisAngle>Prey_a</VAR_TVisAngle>
     </ANGLE-OP></COM-TURN-OPERAND>
    <COM-TURN-OPERAND ind="47">
    <SIGN>-</SIGN>
    <ANGLE-OP ind="50">
     <VAR_TVisAngle>Peer_a</VAR_TVisAngle>
    </ANGLE-OP></COM-TURN-OPERAND>
   </COM-TURN>
  </STM></STML></THEN>
  <NA ind="53"><STML ind="54"><STM ind="55">
   <COM>Exit</COM>
  </STM></STML></NA>
 </IF-THEN-NA></STM></STML></THEN>
 <NA ind="58"><STML ind="59"><STM ind="60">
  <COM>Go_0.75</COM>
 </STM></STML></NA>
</IF-THEN-NA></STM>
</GP>
```

그림 5 최적수 유전자 프로그램의 XML 표현

```
Program Main;
   type
     TDistance =    0..400;
     TVisAngle = -180..180;
     TSpeed    =    0.. 20;
   var
     Speed          : TSpeed;
     Peer_d, Prey_d: TDistance;
     Peer_a, Prey_a: TVisAngle;
     PeerVisible, PreyVisible: Boolean;
  Procedure GP;
    begin
      try if (Prey_a >= -19) then begin
        try if (Prey_d <= 382)then
          Turn(Prey_a-18+Prey_a-Peer_a);
        except Exit;
      end;
      except Go_0.75;
  end;
  begin // main program
```

그림 6 그림 5의 유전자 프로그램에 대한 파스칼 형식의 표현

기 쉽도록 파스칼 형태로 보인다. 여기서 예외처리는 try...except 문장으로 표현된다.

그림 7은 그림 6에 보인 프로그램이 10개의 초기 환경 중의 하나에서 실행되는 과정을 보인다. 시뮬레이션이 시작될 때에는 중앙에 놓여 있던 피식자는 118 단계가 지난 후 포식자에게 잡혔음을 알 수 있다. 여기서 다음과 같은 포식자의 행동이 창발한 것은 주목할 만하다: (a) 에이전트 ②가 맹목적 추격 행동에서 둘러싸기 행동

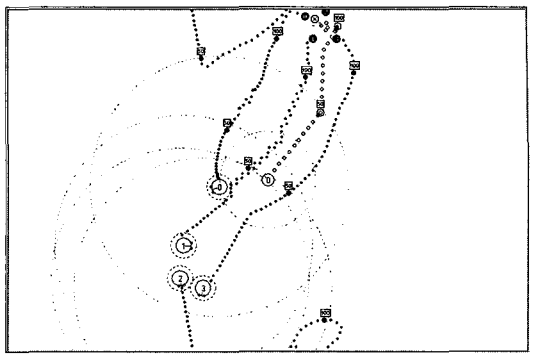


그림 7 그림 5, 6에서 보인 유전자 프로그램의 실행 궤적. 피식자는 118 시뮬레이션 시간 단계에서 잡혔다(위쪽 가운데). 흰색 큰 원과 작은 검은색 원은 각각 포식자의 초기위치와 최종위치를 나타낸다. 흰색 작은 원은 피식자를 나타내는데, 초기에는 중간에 놓인다. 사각형 안의 숫자는 당시의 시간의 경과를 의미한다.

으로 전환한 점(시간 65, 윗부분 가운데에서 다른 포식자들이 감지하자마자) (b) 에이전트 ①이 지그재그 행동을 함으로써 결과적으로 추격 속력을 떨어뜨린 점. 이는 피식자를 포위하려는 의도로 보임(시간 40, 중앙) (c) 에이전트 ⓞ, ③에 의한 둘러싸기(위)와 에이전트 ②가 마지막 단계에서 둘러싼 점.

3.3 계산시간에 대한 분석

그림 8은 병렬 분산 환경에서 XML기반 강건 타입형 유전자 프로그램 구조를 보인다. 개발된 원형은 하나의 인스턴스가 존재하는 GP 관리자와 컴퓨터마다 하나의 인스턴스가 존재하는 평가 서버로 구성된다. 관리자는 DOM 파스트리로 표현되는 유전자 프로그램의 집단을 유지하고, 강건 타입형 프로그램의 특성에 맞도록 DOM 파서 고유의 API를 호출하여 유전자 연산을 수행한다. 관리자는 또 DOM 파서 API를 호출함으로써 유전자 프로그램을 DOM 파스트리에서 XML 형태로 변환하고 이를 라운드 로빈(round robin) 방식의 스케쥴링을 통하여 이용 가능한 평가 서버에 순차적으로 배분한다.

평가 서버는 XML 형태의 유전자 프로그램을 받은 후 DOM 파서 API를 이용하여 XML 형태를 DOM 파스트리 형태로 변환하고, 그 결과 생성되는 DOM 파스트리를 평가 쓰레드에 할당한다. 평가과정은 10개의 서로 다른 초기 환경에 대하여 동일한 유전자 프로그램을 제어구조로 사용하는 포식자들이 하나의 팀을 이루어서 피식자를 잡는 과정을 시뮬레이션 한다. 평가과정이 종료하게 되면, 평가 서버는 그에 해당하는 적합도를 관리자에 보낸다.

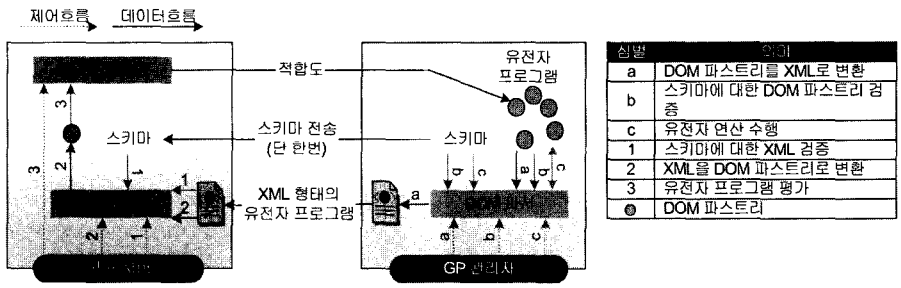


그림 8 병렬 분산 환경에서 XML 기반 강건 타입형 유전자 프로그램의 구조

XML 표현의 계산시간은 다음과 같은 두 가지 관점에서 평가되었다: (a) W3C 표준 XML 스키마를 사용한 강건 타입형 유전자 프로그램의 문법 표현에 의한 계산노력의 감소[9], (b) 강건 타입형 유전자 프로그램의 병렬분산 구현에 의한 계산효율의 향상.

계산노력은 데이타 타입이 존재하지 않는 일반적인 유전자 프로그램, 부분적으로 데이타 타입을 지원하는 유전자 프로그램, 그리고 강건 타입형 유전자 프로그램에 대하여 20번의 독립적인 실행을 통하여 구하여 졌다. 이는 [1]과 마찬가지로 성공에 대한 누적 확률 $p(t)$[10]를 가진 해를 얻기 위해 필요한 유전자 프로그램의 수로 정의된다.

그림 9에서 보이듯이, 성공확률 $p(t)=0.95$에 대하여 강건 타입형 유전자 프로그램은 약 28,000 개의 유전자 프로그램에 대한 평가가 필요하다. 반면, 느슨한 인식 데이타 타입(loose perception data type, LP)[11], 느슨한 행동 데이타 타입(loose action data type, LA)[12], 느슨한 인식-행동 데이타 형(loose perception and action data type, LPA)은 모두 0.95보다 훨씬 작은 값에서 머무르고 있다. LP에서 $p(t)$는 겨우 0.4인데, 이는 20번의 독립적인 실행에서 8번만이 성공적으로 끝났음을 의미한다. LPA[13]의 경우에는 이보다 더 적은 성공확률을 보여주었다. 이 결과는 일반적인 유전자 프로그램보다 탐색공간을 줄인 강건 타입형 유전자 프로그램이 보다 나은 계산노력을 보여줌을 증명한다.

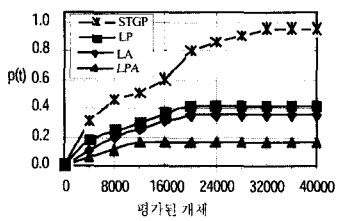


그림 9 강건 타입형 유전자 프로그램과 비 강건 타입형 유전자 프로그램의 계산노력

비록 제안된 방식이 나은 계산노력을 보여주지만, 요구 메모리양 증가 및 분산 환경에서 이주되는 정보의 크기가 증가된다는 단점도 있다. 메모리의 증가는 파스트리에 유지되는 정보가 늘어남에 원인이 있다. 이는 곧 캐시 및 가상 메모리의 실패 확률의 증가로 이어지므로, 이를 회피하기 위해서는 유전자 프로그램의 집단의 크기를 효율적으로 유지가능한 적정한 크기로 해야 한다는 제약점이 발생한다. 하지만 이는 유전자 프로그램의 병렬, 분산적인 구현에 따른 이득에 의하여 상쇄되고 남을 것으로 보인다.

또한 XML 기반 텍스트 표현을 사용하기 때문에 S-표현 등과 비교하여 유전자 프로그램의 크기가 상대적으로 증가한다. 이는 분산 환경에서 하나의 계산노드에서 다른 노드로의 유전자 프로그램의 이주에 더 많은 부담이 발생함을 의미한다. 하지만 유전자 프로그램의 XML 표현은 네트워크 환경에서의 다른 통신요구량보다 상대적으로 적다는 점, 그리고 통신상의 오버헤드는 해당 소프트웨어의 전송지연을 포함한다는 점에서 볼 때 실제의 오버헤드는 데이타 전송지연에 따른 정도의 오버헤드일 것으로 보인다. 실제로 다중 에이전트 시스템에서는 적합도 평가 단계가 시간을 가장 많이 소비하는 단계이기 때문에, 이 논문에서 쓰이는 중간 크기의 보스-작업자 병렬 모델에 있어서의 전체적인 통신 부담(수십 밀리 초에서 수백 밀리 초까지)은 유전자 프로그

9) 가지치기가 된 경우와 안 된 경우 둘의 해의 밀집도가 비슷한 경우 등 몇몇 경우에 있어서는 광대하고 다차원적인 탐색공간의 유지가 가지치기를 통해 축소된 탐색공간보다 효율적일 수도 있다는 주장도 있다.

10) 예를 들어 20번 실행 중 19번을 성공한 경우 $p(t)=\frac{19}{20}=0.95$.

11) IF-THEN 규칙의 결론부의 행동 부분에만 데이타 타입이 존재. 그러므로 조건부분에, 가령 피식자와 다른 포식자와의 거리 등이 무의미한 값(예를 들면, 1000)과 비교될 수 있다.

12) IF-THEN 규칙의 조건부에만 데이타 타입이 존재. 따라서 결론부분에, 가령 Turn() 행동의 매개값로 의미상 각도를 의미하는 값 뿐 아니라 의미상으로 무의미한 값(예를 들면, 거리)도 올 수 있다.

13) IF-THEN 규칙에서 조건부 및 결론부 모두에 데이타 타입이 부재하여, 일반 GP의 특성을 나타낸다.

램 자체의 실행(수초에서 수십 초, 2GHz 펜티엄 4)에 비하여 훨씬 적은 부담이 될 것으로 보인다. 따라서 통신량의 미소한 증가에 따른 통신 오버헤드의 증가보다는, 병렬/분산적 구현에 따른 확장성 및 병렬 실행에 따른 시간적 이득이 크며, 이러한 계산효율의 향상은 $O(n)$(n:평가서버의 수)이다.

4. 결 론

이 논문은 유전자 프로그램을 통한 계통적 학습의 효율적 수행을 위해서 XML 기반 유전자 표현과 역할에 관한 연구결과를 제시하였다. 제안한 접근방식이 기반하고 있는 기본 아이디어에 대한 검증은 포식자-피식자 문제를 다룬 다중 에이전트 시스템의 에이전트의 사회적 행동을 진화하는 유전자 프로그램의 구현을 통하여 검증되었다.

제안된 방법은 다음과 같은 특징이 있다. 첫째, XML 태그는 강건 타입형 유전자 프로그램에서 데이타 형의 관리에 관한 포괄적인 지원을 제공한다. 둘째, W3C 표준 XML 스키마는 강건 타입형 유전자 프로그램의 문법을 표현하는데 대한 포괄적인 방법을 제공한다. 셋째, 유전자 프로그램의 관리 및 변경은 DOM 파서 본래의 API를 사용할 수 있다. 넷째, 사용되는 파서는 OS에 중립성이다. 다섯째, DOM 파서는 언어 중립적이다. 마지막으로, 제안된 유전자 표현은 네트워크 환경에 적합하다.

한편, 제안한 방법은 다음과 같은 기여점이 있다. 첫째, 유전자 프로그램을 소프트웨어 공학하는데 소비되는 시간을 줄임으로써 설계 시간을 단축하는데 기여한다. 둘째, 의미적으로 올바른 유전자 프로그램만을 다룸으로써 탐색공간을 제약하고 결과적으로 계산노력을 줄이는데 대한 포괄적인 방법을 제공한다. 셋째, 이형 분산적인 컴퓨팅 환경에서 컴퓨팅 노드 사이의 유전자 프로그램의 이주에 용이한 XML 형식을 이용하기 때문에 유전자 프로그램의 계산효율을 향상시키기 위한 병렬적 구현 방법을 제공한다.

추후에 보충되어야 할 연구내용은 다음과 같다. XML 형식의 유전자 프로그램 표현을 사용하기 때문에 캐시 및 가상 메모리의 실패확률의 증가가 예상되며, 네트워크상에서의 전송부담이 다소 늘어날 것으로 예상된다. 하지만, 이러한 추가적인 부담은 병렬/분산적 구현에 따른 확장성 및 병렬 실행에 따른 시간적 이득에 비하면 미미할 것으로 보이며, 추후에 이에 대한 보다 정량적인 분석이 필요하다.

결론적으로, 유전자 프로그램의 영역 독립적인 특성으로 인하여 제안한 접근방식은 다양한 문제 영역에서 빠르게 실행될 수 있는 유전자 프로그램을 단기간에 개발하는데 적용될 수 있을 것으로 보인다.

참 고 문 헌

[1] Koza, J. R., Genetic Programming: *On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, MIT Press, 1992.

[2] Morowitz, H. J., *The Emergence of Everything: How the World Became Complex*, Oxford University Press, New York, 2002.

[3] Meeden, L., Kumar, D., "Trends in evolutionary robotics," *Soft Computing for Intelligent Robotic Systems*, Physica-Verlag, New York, NY, pp. 215-233, 1998.

[4] Parunak, H. Van D., Brueckner, S., Fleischer, M., Odell, J., "Co-X: Defining what agents do together," *Proceedings of the AAMAS 2002 Workshop on Teamwork and Coalition Formation*, Bologna, Italy, 2002.

[5] Genetic Algorithms Research and Applications Group, lil-gp Genetic Programming System, URL: http://garage.cps.msu.edu/software/lil-gp/index.html , 1998.

[6] Luke, S., ECJ9: *A Java-based Evolutionary Computation and Genetic Programming Research System*, URL: http://www.cs.umd.edu/projects/plus/-ec/ecj/, 2001.

[7] Haynes, T., Wainwright, R., Sen, S., Schoenefeld, D., *Strongly Typed Genetic Programming in Evolving Cooperation Strategies*, 1997.

[8] Montana, D., "Strongly typed genetic programming," *Evolutionary Computation*, Vol. 3, No. 2, pp. 199-230, 1995.

[9] Holand, J. H., Emergence: *From Chaos to Order*, Cambridge, Perseus Books, 1999.

[10] W3C, Extensible Markup Language (XML) 1.0, Second Edition, W3C Recommendation, URL: http://www.w3.org/TR/REC-xml, 2000.

[11] The Open Group, *Architecture Description Markup Language (ADML): The new XML-based standard for IT architecture interoperability*, URL: http://-www.opengroup.org/tech/architecture/adml/adml_home.htm, 2001.

[12] Zou, Y., Kontogiannis, K., "Towards a portable XML-based source code representation," *Proceedings of ICSE 2001 Workshops of XML Technologies and Software Engineering*, Toronto, Canada, May 15, 2001.

[13] Grosof, B. N., Labrou, Y., "An approach to using XML and a rule-based content language with an agent communication language," *Procs. IJCAI-99 Workshop on Agent Communication Languages, Stockholm*, Sweden, 1999.

[14] Beech, D., Maloney, M., Mendelsohn, N., Thompson. H., "XML schema part 1: Structures," *W3C Recommendation*, 2001, URL: http://www.-

w3.org/TR/2001/REC-xmlschema-1-20010502/.

[15] Biron, P., Malhotra, A., "XML schema part 2: Datatypes," *W3C Recommendation*, 2001, URL: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[16] Milenkovich, M., Robinson, S. H., Knauerhase, R. C., Barkai, D., Garg, S., Tewari, V., Anderson, T. A., Bowman M., "Toward Internet distributed computing," *Computer*, Vol. 36, No. 5, pp. 38-46, 2003.

[17] Haynes, T., Sen, S., "Evolving behavioral strategies in predators and prey," In G. WeiB and S. Sen, editors, *Adaptation and Learning in Multiagent Systems, pp. 113-126, Springer-Verlag*, 1996.

[18] Brooks, R. A., "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, Vol. 1, No. 2, pp. 14-23, 1986.

[19] Tanev, I, Uozumi, T., Ono, K., "DCOM-based parallel distributed implementation of genetic programming," *Parallel and Distributed Computing Practices, special issue on Distributed Object Oriented Systems*, Vol. 3, No. 1, pp. 77-88, 2000.

부록 1

XML 스키마로 표현된 강건 타입형 유전자 프로그램의 문법. 강건 타입형 유전자 프로그램은 포식자-피식자 문제를 다룬 다중 에이전트 시스템에서 포식자의 행동을 진화하는데 사용되었다.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="COM"><xs:restriction base="xs:string">
    <xs:enumeration value="Null"/>
    <xs:enumeration value="Stop"/>
    <xs:enumeration value="Go_0.25"/>
    <xs:enumeration value="Go_0.5"/>
    <xs:enumeration value="Go_0.75"/>
    <xs:enumeration value="Go_1.0"/>
    <xs:enumeration value="Exit"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="CONST_TDistance"><xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="400"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="CONST_TVisAngle"><xs:restriction base="xs:integer">
    <xs:minInclusive value="-180"/>
    <xs:maxInclusive value="+180"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="CONST_TSpeed"><xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="20"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="VAR_TDistance"><xs:restriction base="xs:string">
    <xs:enumeration value="Prey_d"/>
    <xs:enumeration value="Peer_d"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="VAR_TVisAngle"><xs:restriction base="xs:string">
    <xs:enumeration value="Prey_a"/>
    <xs:enumeration value="Peer_a"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="VAR_TSpeed"><xs:restriction base="xs:string">
    <xs:enumeration value="Speed"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="OPER_TDistance"><xs:restriction base="xs:string">
    <xs:enumeration value="GE"/>
    <xs:enumeration value="LE"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="OPER_TVisAngle"><xs:restriction base="xs:string">
    <xs:enumeration value="GE"/>
    <xs:enumeration value="LE"/>
    <xs:enumeration value="WI"/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="OPER_TSpeed"><xs:restriction base="xs:string">
    <xs:enumeration value="GE"/>
    <xs:enumeration value="LE"/>
</xs:restriction></xs:simpleType>
<xs:complexType name="COND_TSpeed"><xs:sequence>
      <xs:element name="VAR_TSpeed" type="VAR_TSpeed"/>
      <xs:element name="OPER_TSpeed" type="OPER_TSpeed"/>
      <xs:element name="CONST_TSpeed" type="CONST_TSpeed"/>
</xs:sequence>
<xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COND_TDistance"><xs:sequence>
      <xs:element name="VAR_TDistance" type="VAR_TDistance"/>
      <xs:element name="OPER_TDistance" type="OPER_TDistance"/>
      <xs:element name="CONST_TDistance" type="CONST_TDistance"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COND_TVisAngle"><xs:sequence>
      <xs:element name="VAR_TVisAngle" type="VAR_TVisAngle"/>
      <xs:element name="OPER_TVisAngle" type="OPER_TVisAngle"/>
      <xs:element name="CONST_TVisAngle" type="CONST_TVisAngle"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:simpleType name="OPER_TBool"><xs:restriction base="xs:string">
    <xs:enumeration value="not"/>
    <xs:enumeration value="."/>
</xs:restriction></xs:simpleType>
<xs:simpleType name="VAR_TBool"><xs:restriction base="xs:string">
    <xs:enumeration value="PeerVisible"/>
    <xs:enumeration value="PreyVisible"/>
</xs:restriction></xs:simpleType>
<xs:complexType name="COND_TBool"><xs:sequence>
      <xs:element name="OPER_TBool" type="OPER_TBool"/>
      <xs:element name="VAR_TBool" type="VAR_TBool"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COND-THEN"><xs:choice>
      <xs:element name="COND_TSpeed" type="COND_TSpeed"/>
      <xs:element name="COND_TBool" type="COND_TBool"/>
    </xs:choice>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
  <xs:attribute name="age" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COND-THEN-NA"><xs:choice>
      <xs:element name="COND_TDistance" type="COND_TDistance"/>
      <xs:element name="COND_TVisAngle" type="COND_TVisAngle"/>
    </xs:choice>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="THEN"><xs:sequence>
      <xs:element name="STML" type="STML"/>
  </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="NA"><xs:sequence>
      <xs:element name="STML" type="STML"/>
  </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="IF-THEN"><xs:sequence>
      <xs:element name="COND-THEN" type="COND-THEN"/>
      <xs:element name="THEN" type="THEN"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="IF-THEN-NA"><xs:sequence>
      <xs:element name="COND-THEN-NA" type="COND-THEN-NA"/>
      <xs:element name="THEN" type="THEN"/>
      <xs:element name="NA" type="NA"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:simpleType name="SIGN"><xs:restriction base="xs:string">
    <xs:enumeration value="+"/>
    <xs:enumeration value="-"/>
</xs:restriction></xs:simpleType>
<xs:complexType name="ANGLE-OP">
    <xs:choice>
      <xs:element name="VAR_TVisAngle" type="VAR_TVisAngle"/>
    </xs:choice>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COM-TURN-OPERAND"><xs:sequence>
      <xs:element name="SIGN" type="SIGN"/>
      <xs:element name="ANGLE-OP" type="ANGLE-OP"/>
    </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="COM-TURN"><xs:sequence>
      <xs:element name="COM-TURN-OPERAND" maxOccurs="unbounded"
      type="COM-TURN-OPERAND"/>
  </xs:sequence>
  <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="STM"><xs:choice>
      <xs:element name="COM" type="COM"/>
      <xs:element name="COM-TURN" type="COM-TURN"/>
```

```
        <xs:element name="IF-THEN" type="IF-THEN"/>
        <xs:element name="IF-THEN-NA" type="IF-THEN-NA"/>
      </xs:choice>
    <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:complexType name="STML"><xs:sequence>
        <xs:element name="STM" maxOccurs="unbounded"  type="STM"/>
    </xs:sequence>
    <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType>
<xs:element name="GP"><xs:complexType><xs:sequence>
        <xs:element name="STM" type="STM"/>
      </xs:sequence>
    <xs:attribute name="ind" type="xs:integer" use="optional"/>
</xs:complexType></xs:element>
</xs:schema>
```

이 승 익

1997년 8월 연세대학교 컴퓨터과학과 석사. 2002년 2월 연세대학교 컴퓨터과학과 공학박사. 2002년7 월~2003년 12월 일본 국제전기통신기초기술연구소 객원연구원. 2004년 2월~현재 한국전자통신연구원 선임연구원. 관심분야는 로보틱스, 진화연산, 인공생명, 패턴인식, 지능제어

Ivan Tanev

Dr. Tanev earned M.S. (with honors) and Ph.D. degrees from Saint-Petersburg State Electrotechnical University, Russia in 1987 and 1993 respectively, and Dr.Eng. degree from Muroran Institute of Technology, Japan in 2001. He has been with the Bulgarian Space Research Institute (1987), Bulgarian Central Institute of Computer Engineering and Computer Technologies. (1988-1989), Bulgarian National Electricity Company (1994-1997), Synthetic Planning Industry Co.Ltd., Japan (2001-2002), and ATR Human Information Science Laboratories (2002-2004), Japan. Since April 2004 he has been a visiting researcher at the ATR Network Informatics Laboratories, and a lecturer at the Doshisha University, Japan. Dr. Tanev's research interests include evolutionary computations, multi-agent systems and socioinformatics.

부록 2

Backus Naur Form(BNF)의 XML 스키마로의 변환규칙

BNF	XML 스키마
<L>::=<RN1>·<RN2>···<RNn>	<xs:complexType name="L"> <xs:sequence> <xs:element name="RN1" type=" RN1"/> <xs:element name="RN2" type=" RN2"/> ... <xs:element name="RNn" type=" RNn"/> </xs:sequence> </xs:complexType>
<L>::=<RN1>|<RN2>|···|<RNn>	<xs:complexType name="L"> <xs: choice > <xs:element name="RN1" type=" RN1"/> <xs:element name="RN2" type=" RN2"/> ... <xs:element name="RNn" type=" RNn"/> </xs: choice > </xs:complexType>
<L>::=<RN1>|<RN2>|···{<RNi>}| ···|<RNn>	<xs:complexType name="L"><xs:sequence> <xs:element name="RN1" type=" RN1"/> <xs:element name="RN2" type=" RN2"/> ... <xs:element name="Ri" maxOccurs="unbounded" type="Ri"/> ... <xs:element name="RNn" type=" RNn"/> </xs:sequence> </xs:complexType>
<L>::=<RT1>|<RT2>|···|<RTn>	<xs:simpleType name="L"> <xs:restriction base="xs:string"> <xs:enumeration value="RT1"/> <xs:enumeration value="RT2"/> ... <xs:enumeration value="RTn"/> </xs:restriction> </xs:simpleType>
<L>::= "a..b", where a and b are integers, a<b	<xs:simpleType name="L"> <xs:restriction base="xs:integer"> <xs:minInclusive value="a"/> <xs:maxInclusive value="b"/> </xs:restriction> </xs:simpleType>

Katsunori Shimohara

Katsunori Shimohara, Dr. Eng. is Director, Network Informatics Laboratories, Advanced Telecommunications Research Institute (ATR) International, and Guest Professor at the Graduate School of Informatics, Kyoto University, both in Kyoto, Japan. His research interests include human communication mechanisms, evolutionary systems & artificial life, artificial brains and emotions, genome informatics, and human-system interactions.