Programming Dynamic Networks of Mobile and Stationary Devices

Ulrich (Uli) Kremer

Computer Science
Rutgers University
Programming Dynamic Networks of Mobile and Stationary Devices

Pradip Hari
Liviu Iftode
Ulrich (Uli) Kremer
Yang Ni
Desiree Ottoni
Adrian Stere
Hans Christian Woithe

Kevin Ko
Emmanouil Koukoumidis
Margaret Martonosi
Li Shiuan Peh
Pei Zhang

David Aragon
Scott Glenn
Josh Kohut
Oscar Schofield

Computer Science
Rutgers University

Electrical Engineering
Princeton University

Marine Science
Rutgers University
A New Computing Platform?

Ad-hoc/dynamic networks make sharing and cooperation possible among small devices.

Related Work:
- Network of Workstations (NOW)
- Peer-to-Peer (P2P) systems
Applications for Dynamic Networks

- **Collaborative computing.**
  - Location-sensitive
  - Exploiting multiple services
  - In-network processing
  - Location resilient services

- **Sensor data collection**
  - Location-sensitive
  - QoR vs. resource consumption tradeoff
  - In-network processing
  - Streamed data
  - Time constraint

- **Augmented-reality games.**
  - Location-sensitive
  - Physical environment attached with virtual information
An AMBER Alert Emergency System

Initialization Phase
- dynamically deploy problem-specific image understanding software
- space resilient software coverage
- alert AMBER participants; share initial search information

Search Phase
- periodic acquisition of images; image analysis by nearby nodes (in-network processing)
- reporting back results to emergency center
- sharing information with nearby AMBER participants

- periodic exchange of information between nearby AMBER participants
- report back results to emergency center
Talk Outline

• Motivation

• SpatialViews
  • Language
  • Implementation
  • Applications

• Sarana
  • Research challenges
  • Preliminary results

• Slocum Gliders
SpatialViews Project

A programming model for ad-hoc/dynamic networks

- Dynamic integration of services
  - Deploy, discover and use services dynamically
- Location-awareness
  - Location specification at the language level
  - Enable location-aware optimizations
- Quality of Result (QoR) specification
  - Result of different qualities (number of visited nodes) with respect to space and time
- Security and Privacy
  - No language support

http://www.cs.rutgers.edu/spatialviews
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n > 0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}

• A spatial view defines a set of interesting nodes
• “Migration”-based iteration lazily discovers and visits the interesting nodes.
1. Virtual nodes are unique in terms of space and time (space / time granularity)

2. Physical nodes can “instantiate” multiple virtual nodes
Spatial View

Spatial View
- A dynamic collection of nodes that provide interesting services and are located in an interesting space.
- A service is named using an interface.
  Service implementation is either system-provided, or user-installed.
- A space is represented as an object of Space class or its subclasses.

Examples

```
spatialview sv1 = Camera @ ConferenceRoom
spatialview sv2 =
    Camera @ new Circle(currentLoc,10)
spatialview sv3 = LightSensor @ CampusB % 300
```
An iterator (visiteach) instantiates a spatial view, by

- Discovering interesting nodes, with the interesting services and located in the interesting space, and
- Migrating execution to those nodes, running the specified code (which uses the service) on the nodes, and returning to the injecting node (i.e. the node where the iteration started).
- Iterators can be nested.

Examples

```plaintext
visiteach x : sv1 { ... }
visiteach x : sv2 every 3 within 600 { ... }
visiteach x : sv3 every 5 forever { ... }
```
Routing/Replication/Parallelization

- Hidden in the iteration, transparent to a user.
- Selected by user through compiler options.
- Routing implemented in the runtime library.
**Iteration Approaches**

**Simple Approaches:** try to visit all nodes in the interesting space

- Serial [LCPC’03]
- Parallel [LCR’04]
  - Replicated (Independent)
  - Flooding (Cooperative)
  - Tree-based (Cooperative)

**Geographic Approaches:** selectively visit nodes in the interesting space based on their locations [PLDI’05]

- Geographic Serial
- Geographic Flooding
Example: Aggregating Sensor Data

import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction int n = 0;

        visiteach x : v {
            s += x.read(); n++;
        }

        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
Example: Aggregating Sensor Data

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Evenly distribution of nodes:
- Divide the space into subspaces of size $\Delta s \times \Delta s$ (e.g. $\Delta s=320m$)
- Visit at most one interesting node from a subspace.
Example: Aggregating Sensor Data

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding:
- Propagate computation into the network down the hierarchy of the space.
Example: Aggregating Sensor Data

```
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction int n = 0;

        visiteach x : v {
            s += x.read(); n++;
        }

        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding:
- Propagate computation into the network down the hierarchy of the space.
Example: Aggregating Sensor Data

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding:
- Propagate computation into the network down the hierarchy of the space.
Example: Aggregating Sensor Data

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding:
- Propagate computation into the network down the hierarchy of the space.
Example: Aggregating Sensor Data

import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}

Geographic Flooding:

• Propagate computation into the network down the hierarchy of the space.
Example: Aggregating Sensor Data

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding:

- Propagate computation into the network down the hierarchy of the space.
**Memory Model**

- Each node has a separate memory space in the underlying system.
- A “Shared Memory” with **RESTRICTIONS** at the language level.
  - A well-defined semantics for both serial and parallel iterations (“FORALL”).
  - A single iteration step is an atomic transaction.
  - **Program Variables**: *Local Variables* and *Reduction Variables*.
    - “Shared” by cloned threads of a program across the network.
    - Local Variables are read-only in a nested iterator.
    - Reduction Variables are write-only with a given reduction operation.
  - **Service Variables**
    - Stay on individual nodes, not accessible in a nested iterator.
    - Shared by same or different programs executing on the same node.
Each node has a separate memory space in the underlying system.

A “Shared Memory” with RESTRICTIONS at the language level.

- A well-defined semantics for both serial and parallel iterations (“FORALL”).
- A single iteration step is an atomic transaction.
- **Program Variables**: Local Variables and Reduction Variables.
  - “Shared” by cloned threads of a program across the network.
  - Local Variables are read-only in a nested iterator.
  - Reduction Variables are write-only with a given reduction operation.
- **Service Variables**
  - Stay on individual nodes, not accessible in a nested iterator.
  - Shared by same or different programs executing on the same node.
Memory Model

Each node has a separate memory space in the underlying system.

A “Shared Memory” with RESTRICTIONS at the language level.

- A well-defined semantics for both serial and parallel iterations (“FORALL”).
- A single iteration step is an atomic transaction.
- **Program Variables**: Local Variables and Reduction Variables.
  - “Shared” by cloned threads of a program across the network.
  - Local Variables are read-only in a nested iterator.
  - Reduction Variables are write-only with a given reduction operation.

- **Service Variables**
  - Stay on individual nodes, not accessible in a nested iterator.
  - Shared by same or different programs executing on the same node.
SpatialViews Prototype

An extension to J2ME 1.0, including:

• A customized Java Compiler (javac) and runtime library, supporting
  - Parallelization using different iteration approaches with in-network reduction or aggregation;
  - Static checking for memory restrictions;
  - Transparent thread migration.
• A customized Kilobyte Virtual Machine (KVM) supporting explicit thread migration (SmartMessages).
• A development environment with a mobile network simulator/emulator.
SpatialViews Compiler / Runtime System

SpatialViews Program → SpatialViews Compiler → Java Bytecode → SmartMessages KVM

Customized JDK 1.3.1 from SUN

Parser → Intermediate Representation → Bytecode Generator

Translator
Smart Messages

Cristian Borcea, Porlin Kang, Liviu Iftode
Rutgers

Smart Messages Node Architecture

Smart Messages Platform

Admission Manager

SM Ready Queue

Virtual Machine

Tag Space

Code Cache

Operating System & I/O

SM Network

SM Network
Compiler Generated Code for Sensor Data Aggregation Example

public class AverageLighting {

    public class SV0 extends GeoFlooding {
        public SV0() {
            super();
        }

        public float s = 0;
        public int n = 0;

        public void reset() {
            super.reset();
            s = 0;
            n = 0;
        }

        public void userTask() {
            final Sensor x;
            x = (Sensor) getServiceProvider();
            if (x == null) return;
            s += x.getValue();
            n += 1;
            writeResidentGlobalVariable("s", new java.lang.Float(0));
            writeResidentGlobalVariable("n", new java.lang.Integer(0));
        }

        public void merge() {
            try {
                s += ((java.lang.Float) readResidentGlobalVariable("s")).floatValue();
            } catch (Exception dummy) {
            }
            writeResidentGlobalVariable("s", new java.lang.Float(s));
            try {
                n += ((java.lang.Integer) readResidentGlobalVariable("n")).intValue();
            } catch (Exception dummy) {
            }
            writeResidentGlobalVariable("n", new java.lang.Integer(n));
        }

        public void writeBack() {
            try {
                s += ((java.lang.Float) readResidentGlobalVariable("s")).floatValue();
            } catch (Exception dummy) {
            }
            writeResidentGlobalVariable("s", new java.lang.Float(0));
            try {
                n += ((java.lang.Integer) readResidentGlobalVariable("n")).intValue();
            } catch (Exception dummy) {
            }
            writeResidentGlobalVariable("n", new java.lang.Integer(0));
        }

        public AverageLighting() {
            super();
        }

        public static void main(String[] args) throws java.sm.SMCodeCacheException {
            String[] codeBricks = new String[] {"GeoFlooding", "AverageLighting", "AverageLighting$SV0"};
            Object[] dataBricks = new Object[] {new SV0()};
            java.sm.SmartMessage.createSMFromFiles(codeBricks, dataBricks);
            SV0 v = (SV0) java.sm.SmartMessage.getDataBrick(0);
            v.setService("Sensor");
            v.setSpace(SpaceDefs.CampusB);
            v.setDeltaS(320);
            v.reset();
            v.iterate();
            v = (SV0) java.sm.SmartMessage.getDataBrick(0);
            s += v.s;
            n += v.n;
            catch (Exception dummy) {
            }
            System.err.println(dummy);
            System.exit(-1);
            if (n > 0) System.out.println(Float.toString(s / n));
        }
    }
}

• Bytecode is generated. Shown here is the equivalent source code.
• Library code is not shown.
Aggregation of Sensor Data (Cont.)

• Compaq iPAQs
• 802.11b (ad-hoc)

Energy Measurement Setup
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;

        visiteach x : v {
            s += x.read(); n++;
        }

        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}

- 12 iPAQs
- 625m x 625m space (fake)
- Signal range: 250m (unit disk model)
- 802.11b in ad-hoc mode with static configured topology
- space granularity: 320m
Aggregation of Sensor Data (Cont.)

import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}

Geographic Serial Iteration

• 12 iPAQs
• 625m x 625m space (fake)
• Signal range: 250m (unit disk model)
• 802.11b in ad-hoc mode with static configured topology
• space granularity: 320m

6 nodes visited.
Aggregation of Sensor Data (Cont.)

```java
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB % 320;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }

        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
```

Geographic Flooding Iteration

- 12 iPAQs
- 625m x 625m space (fake)
- Signal range: 250m (unit disk model)
- 802.11b in ad-hoc mode with static configured topology
- space granularity: 320m

6 nodes visited.
public class DedicatedLocationService {
    public void static main(String[] args) {
        float dt=Float.valueOf(args[0]).floatValue();
        spatialview sv = @ Hallway;
        visiteach x : sv every dt forever {
            CricketLocationService ls=
                new CricketLocationService();
            Location loc=ls.currentLocation();
            register(dt) EagerLocationService(loc);
        }
    }
}

public class EagerLocationService
    implements LocationService {
    Location l;
    public EagerLocationService(l) { this.l=l; }
    public Location currentLocation() { returns l; }
}

public interface LocationService {
    public Location currentLocation();
}
User-installed Location Service (Cont.)

• Cricket takes up to 4 secs to acquire location.

• Application exec. time = 8 secs using lazy cricket location service.

• 6 iPAQs using simple flooding.

Sensor Application using Eager Location Service
Programming Environment

Emulation/Simulation

- replay capabilities of real traces
- scenario explorations adding/deleting nodes
  motion models interaction models

AIS data provided by Scott Glen (Rutgers)
Current SpatialViews Applications

Marcin Pawelek, Shaila Musharoff, Viktor Raskin, William Fong
Rutgers, 2005

Four Applications.

• **Real-time bus schedule**: send queries over an ad-hoc network to get real-time bus locations to minimize waiting time.
  - Straight forward query
  - Caching and sharing query results using user-installed service
  - Bundled query among users who are close to each other

• **Evacuation coordination**: coordinate evacuation of people from a building by using their handheld devices.
Current SpatialViews Applications

- **AR Monopoly game**: “trade” real properties in a real city, e.g., Manhattan.

- **AR Pong game**: run in real world to bounce a “virtual” ball.
Talk Outline

• Motivation

• SpatialViews
  • Language
  • Implementation
  • Applications

• Sarana
  • Research challenges
  • Preliminary results

• Slocum Gliders
Sarana Project

Space-Aware and Resource-Aware Network Architecture

Research Challenges (incomplete list!)

• Extension of language expressiveness
  - Space/time iterations, streaming, space resilient services, space definitions, …

• Resource cost model
  - Sharing (economic) incentive, accountability
  - Compile-time, “trace-based” (memoization), run-time

• QoR model

• Sublanguage for time/QoR/resource cost tradeoffs
  - Communicate tradeoffs to compiler and runtime system
  - QoR feedback to program

• Multi-level optimizations to best match application needs and available system resources
QoR: obtain \((\text{AmberAlertParticipants} / \text{matchingImages} = 10)\)
Selected Topics – Cost Model and QoR

Breadth-first

100 credits

100/n credits for each camera node

\textbf{visiteach} camera

\textbf{visiteach} Amber Alert participant

QoR: obtain \((\text{AmberAlertParticipants} / \text{matchingImages} = 10)\)
Selected Topics – Cost Model and QoR

Depth-first

100 credits

visiteach camera

visiteach Amber Alert participant

10 nodes

“running sum” of credits

QoR: obtain \((\text{AmberAlertParticipants} / \text{matchingImages} = 10)\)
Selected Topics – Cost Model and QoR

Two Phase Probing

100 credits

visiteach camera

visiteach Amber Alert participant

QoR: obtain \((\text{AmberAlertParticipants} / \text{matchingImages} = 10)\)
import SpaceDefs.Rutgers;

public class AverageLighting {
    public static void main(String[] args) {
        spatialview v = LightSensor @ Rutgers.CampusB;
        sumreduction float s = 0;
        sumreduction int n = 0;
        visiteach x : v {
            s += x.read(); n++;
        }
        if (n>0)
            System.out.println(Float.toString(s/n));
        else
            System.out.println("No sensor found.");
    }
}
Selected Topics – Spatial QoR

Initial Approach

• QoR information is computed for each subspace (320×320)
• QoR information can be represented as a bitvector
• compiler can implement bitvector interpretation using quad trees and geographic routing;
  bitvector compression possible

\[
\begin{array}{cc}
    a & b \\
    c & d \\
\end{array} \quad \Rightarrow a, b, c, d
\]

spatial resolution = 320
counting granularity:
found a representative node in subspace

visited spaces

0 (no)
1 (yes)
Sarana Runtime System Architecture

- distributed directory structure (yellow pages)
- push/pull strategies for code distribution
- location-aware code caching
Related Work

Programming models and tools for ad-hoc (sensor) networks
• SP, Jini, nesC, TAG, TinyDB, Hood, Abstract Regions, Regiment (Region Streams), Kairos, Pleiades, SensorWare, direct diffusion, SNAFU

Sensor network OS and middleware
• TinyOS, maté, Impala, etc.

Ad-hoc network routing
• DSR, AODV, GPRS, LAR, GLS, GeoCast, Mobicast, etc.

Migratory execution
• D’Agent, JAgl et, Obliq, etc.

Pervasive computing systems, which are context-aware and adaptive to dynamic networks:
• Pushpin computing (MIT), solar (Dartmouth), CANS (NYU), etc.
Talk Outline

• Motivation

• SpatialViews
  • Language
  • Implementation
  • Applications

• Sarana
  • Research challenges
  • Preliminary results

• Slocum Gliders
Slocum Gliders

Webb Research, Falmouth, MA
Command and Control: 38 Missions In the Last Year

- 11,312 km flown
- 97,633 casts
- 238 in-water calendar days
- 606 glider days

Source: Scott Glenn
Marine Sc.
Rutgers
New applications need
- More sensors
- Collections of larger data sets
- Onboard processing
- Swarming
- Communication between gliders
- Better location services

Hardware and software capabilities of gliders have to be enhanced through a new programming architecture running on new hardware.
**Current Computer Systems**

- **Payload bay (3 - 5kg):** space for our **new science computer**

- **Two computer systems**
  - **science computer**
  - **flight controller**
  - connected through a **serial link** (RS232)

- Both are 16MHz Motorola 68338 based single board systems, 1MB flash, 512KB SRAM, 8KB virtual EEPROM
Current Programming Model

write mission using existing behaviors

behavior: abend
...  
behavior: surface
...  
behavior: set_heading
  b_arg: use_heading(bool) 4
  b_arg: heading_value(X) -0.46
  b_arg: start_when(enum) 0
  b_arg: stop_when(enum) 5  
...  
behavior: dive_to
  b_arg: target_depth(m) 100
  b_arg: start_when(enum) 4
  b_arg: use_pitch(enum) 3
  b_arg: pitch_value(X) -0.5235
...  
behavior: prepare_to_dive
...  

transfer mission file to glider

parse and execute mission on the glider
Increasing priority

Current Programming Model

Layered Control
- select final command
- new selection every 4 seconds

internal data structure

- c_stack
  - head
  - tail
  - count = 5

command
- abend
  - next
  - prev

command
- surface
  - next
  - prev

command
- rubeh
  - next
  - prev

command
- yo
  - next
  - prev

command
- prepare
  - next
  - prev
Goal

Design new programming architecture for gliders that enables

- In-flight behavior changes based on sensor readings
- Scientists/users to express their applications at a higher level of abstraction
- Software-based safety checks of applications
- Portability across heterogeneous glider fleets

Our Design Philosophy

- Try to be as non-intrusive as possible, i.e., integrate new programming architecture with existing system
- Enhance existing safety checks rather than replacing them
- Build a working prototype that can phase-in new capabilities
Goal: In-flight Behavior Change

write mission with a new behavior that uses existing behaviors as sub-behaviors

behavior: abend
...
behavior: surface
    b_arg: start_when(enum) 1
...
behavior: rubeh
    b_arg: start_when(sec) 240.0
    b_arg: end_when(sec) 360.0
...
behavior: yo
    # updwn_idle
    b_arg: start_when(enum) 4
    # Number of dive/climbs to perform
    b_arg: num_half_cycles_to_do(nodim) 6
...
behavior: prepare_to_dive
    b_arg: start_when(enum) 0
...

transfer mission file to glider
Goal: In-flight Behavior Change

Layered Control
- select final command
- new selection every 4 seconds

Our behavior dynamically generates sub-behaviors as need
Our New Programming Architecture

New science computer
- Linux box
- 5W power (average)
- between flight controller and “old” science

New domain specific language and compiler/ interpreter
- expressive
- safety checks
- optimizations

Interpreter
Interprets mission and sets values in sensor array to produce desired behavior. ex. Set dive_to_flag to 1.

... while (glider.sensor.depth < 50) {
    glider.dive(-0.454)
    if (glider.sensor.temp > 10)
        break;
}
...

new science computer

flight controller

science computer
Our New Programming Architecture

- Behavior: abend
- Behavior: vm
- Behavior: prep

Program in domain specific language:

```c
... while (glider.sensor.depth < 50) {
    glider.dive(-0.454)
    if (glider.sensor.temp > 10)
        break;
} ...
```

Sensor array:

- `dive_to_flag`
- `climb_to_flag`
- `dive_to_arg1`
- `dive_to_arg2`

Interpreter:
Interprets mission and sets values in sensor array to produce desired behavior. ex. Set `dive_to_flag` to 1.

New science computer