Archetype-Based Design: Sensor Network Programming for Application Experts, Not Just Programming Experts

Lan S. Bai†, Robert P. Dick†, Peter A. Dinda‡

†University of Michigan ‡ Northwestern University

Outline

1. Introduction
2. Archetype-based design
3. WASP: an archetype-specific programming language
4. User study

Motivation

Most sensor network needs from application experts.

- E.g., civil engineers, biologists, geologists, and farmers.

All existing applications are implemented in collaboration with embedded system experts.

Application experts generally are novice programmers.

Even basic sensor network design is difficult for them.

Hire embedded system experts or give up.

Disadvantages: cost and disconnection.

Past work

General-purpose, node-level languages.

- NesC, TinyScript, BASIC.

Macroprogramming languages.

- TinyDB, SwissQM, Regiment, Pleiades, ATaG.

Application-specific languages.

- NETSHM for structural health monitoring.

Patterned after LabVIEW or Excel.

Outline

1. Introduction
2. Archetype-based design
3. WASP: an archetype-specific programming language
4. User study

Our idea

Goal: Make designing a substantial sensor network so easy that those without programming experience can do it.

Archetype-specific programming languages.

- Divide design space into regions defined by shared language feature requirements.
- One specialized language for one archetype.
- Program template (examples with annotations and parameters).
- This keeps each language simple and easy to learn.

User-driven design: test the influence of language on correctness and implementation time.

Ours is the first project to evaluate this.

Survey of existing wireless sensor network deployments

23 deployments studied (FireWxNet [Hartung], Golden gate bridge [Kim], etc.).

All developed in collaboration with embedded system experts.

What's the pure application logic?

17 application characteristics identified, e.g., mobility, network lifetime.

8 affect language complexity.

Archetype classification

Identified eight high-impact characteristics

- Mobility
- Initiation of sampling (periodic, event-driven, or both)
- Initiation of data transmission (periodic, event-driven, or both)
- Actuation: triggers events?
- Interactivity: respond to commands during operation?
- Data interpretation: in-network data processing?
- Data aggregation: should data be aggregated across multiple sensor nodes?
- Node homogeneity

Clustering results

Chose to use automated technique.

Used k-means to avoid human bias.

Arch.	Size	Mobil.	Samp.	Data	Actuat.	Interac.	Data	Data	Homo-
				trans.			interp.	agg.	geneous
1	7	stat.	per.	per.	N	N	*	*	Y
2	6	stat.	*	event	N	*	Y	*	Y
3	4	mobile	per.	*	*	N	*	*	Y
4	3	mobile	per.	*	*	Y	*	N	N
5	1	stat.	hybrid	hybrid	N	Y	Y	Y	Y
6	1	stat.	event	hybrid	Y	Y	N	Y	Y
7	1	mobile	per.	event	N	N	Y	Y	N

Clustering results

Chose to use automated technique.

Used k-means to avoid human bias.

Arch.	Size	Mobil.	Samp.	Data	Actuat.	Interac.	Data	Data	Homo-
				trans.			interp.	agg.	geneous
1	7	stat.	per.	per.	N	N	*	*	Y
2	6	stat.	*	event	N	*	Y	*	Y
3	4	mobile	per.	*	*	N	*	*	Y
4	3	mobile	per.	*	*	Y	*	N	N
5	1	stat.	hybrid	hybrid	N	Y	Y	Y	Y
6	1	stat.	event	hybrid	Y	Y	N	Y	Y
7	1	mobile	per.	event	N	N	Y	Y	N

Outline

1. Introduction
2. Archetype-based design
3. WASP: an archetype-specific programming language
4. User study

Designed language for most widely encountered archetype.

WASP: Wireless sensor network Archetype-Specific Programming language.

Node-level code segment specifies sampling and local data processing.

Operations apply to time series data that are local to a single node.

Network-level code segment specifies data filtering, aggregation, transmission through network.

Operations apply to most recent data from all the nodes in the network.

Separation of concerns: node-level + network-level.

Example application

Sample temperature every 2 seconds from all the nodes in the network. Transmit the node identification numbers and the most recent temperature readings from nodes where the current temperature increased by more than 10% during the last 10 seconds.

Used in our user study as Task 3.

WASP code for example application

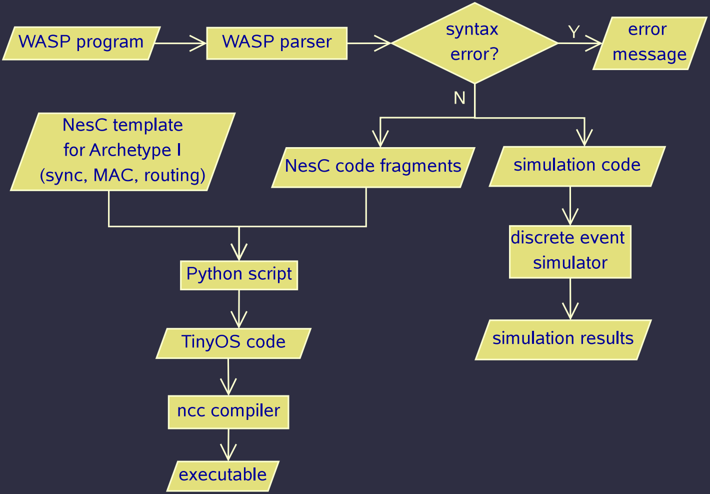
```
local:
sample temperature every 2 sec into mytemp
mintemp = min_time(mytemp[0:4]) every 2 sec
thresh = mintemp * 1.1 every 2 sec

network:
collect nodeid, mytemp
where mytemp > thresh
```

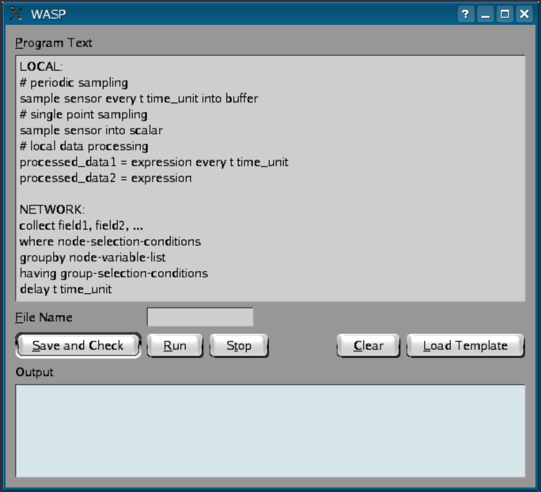
Lines of code for example applications in different languages

Language	T1	T2	T3
WASP	5	7	7
TinySQL	3	4	9
TinyTemplate	49	66	55
TinyScript	49	66	55
NesC	141	453	384

WASP implementation



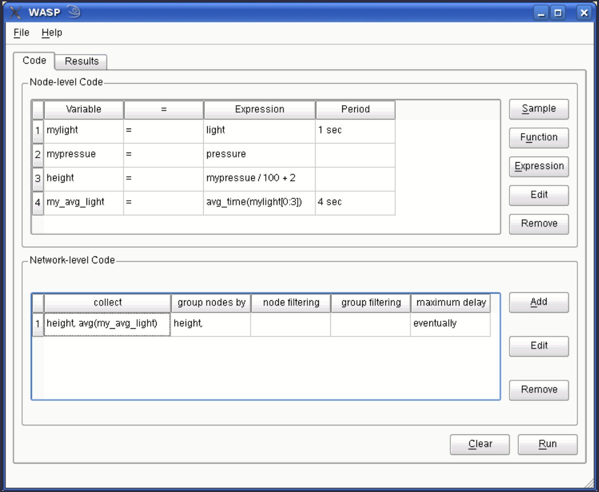
WASP development environment



```
LOCAL:
# periodic sampling
sample sensor every t time_unit into buffer
# single point sampling
sample sensor into scalar
# local data processing
processed_data1 = expression every t time_unit
processed_data2 = expression

NETWORK:
collect field1, field2, ...
where node-selection-conditions
groupby node-variable-list
having group-selection-conditions
delay t time_unit
```

WASP2 development environment



	Variable	=	Expression	Period
1	mylight	=	light	1 sec
2	mypressue	=	pressure	
3	height	=	mypressue / 100 + 2	
4	my_avg_light	=	avg_time(mylight[0:3])	4 sec

	collect	group nodes by	node filtering	group filtering	maximum delay
1	height, avg(my_avg_light)	height,			eventually

Outline

1. Introduction
2. Archetype-based design
3. WASP: an archetype-specific programming language
4. User study

User study I

Goal

- Evaluate usability of WASP and four alternatives.
- Impact of specialized languages, programming template, programming model on programmer productivity?

Protocol

- Test subjects: 28 novice programmers from various fields.
- Three tasks representative of Archetype 1.
- Randomly assign one language and two tasks to each user.
- 30 minutes for tutorial, 40 minutes for each task.
- Functional simulation for user to check correctness.
- Users provide feedbacks on tutorial, language, etc.

User study II

Demographics of test subjects

- 12/28 no programming experience.
- 3/28 wrote at most 500 lines of C++/C/Matlab program.
- 12/28 some experience with C++/C/Matlab/Fortran.
- Researchers in biomedical engineering, civil engineering, chemistry, business and many other fields.

User study III

Languages

- WASP (Our archetype-specific language).
- TinyScript (General-purpose, node-level, event-driven).
- TinyTemplate (Archetype-specific TinyScript with template).
- SwissQM (SQL-like, graphic interface for composing queries).
- TinySQL (Language for TinyDB, SQL-like).

User study results

Language	Success rate			Develop time (min)		
	T1	T2	T3	T1	T2	T3
SwissQM	3/3	3/3	N.A.*	5.7	11.3	N.A.
WASP	2/2	2/4	2/4	16	31	29.5
TinySQL	3/4	2/3	0/3	17.7	27.5	N.A.
TinyTemplate	1/4	0/3	0/3	34	N.A.	N.A.
TinyScript	0/3	0/3	0/4	N.A.	N.A.	N.A.
WASP2	3/3	3/4	2/3	3	9.7	23.5

* Does not support temporal queries.

- WASP and TinySQL are close for T1 and T2, but WASP is easier for T3.
- TinyTemplate, TinyScript: low success rates for archetype.
- Average development time of WASP2 is 47% of WASP.

User study results

Language	Success rate			Develop time (min)		
	T1	T2	T3	T1	T2	T3
SwissQM	3/3	3/3	N.A.*	5.7	11.3	N.A.
WASP	2/2	2/4	2/4	16	31	29.5
TinySQL	3/4	2/3	0/3	17.7	27.5	N.A.
TinyTemplate	1/4	0/3	0/3	34	N.A.	N.A.
TinyScript	0/3	0/3	0/4	N.A.	N.A.	N.A.
WASP2	3/3	3/4	2/3	3	9.7	23.5

* Does not support temporal queries.

- WASP and TinySQL are close for T1 and T2, but WASP is easier for T3.
- TinyTemplate, TinyScript: low success rates for archetype.
- Average development time of WASP2 is 47% of WASP.

User study results

Language	Success rate			Develop time (min)		
	T1	T2	T3	T1	T2	T3
SwissQM	3/3	3/3	N.A.*	5.7	11.3	N.A.
WASP	2/2	2/4	2/4	16	31	29.5
TinySQL	3/4	2/3	0/3	17.7	27.5	N.A.
TinyTemplate	1/4	0/3	0/3	34	N.A.	N.A.
TinyScript	0/3	0/3	0/4	N.A.	N.A.	N.A.
WASP2	3/3	3/4	2/3	3	9.7	23.5

* Does not support temporal queries.

- WASP and TinySQL are close for T1 and T2, but WASP is easier for T3.
- TinyTemplate, TinyScript: low success rates for archetype.
- Average development time of WASP2 is 47% of WASP.

User study results

Language	Success rate			Develop time (min)		
	T1	T2	T3	T1	T2	T3
SwissQM	3/3	3/3	N.A.*	5.7	11.3	N.A.
WASP	2/2	2/4	2/4	16	31	29.5
TinySQL	3/4	2/3	0/3	17.7	27.5	N.A.
TinyTemplate	1/4	0/3	0/3	34	N.A.	N.A.
TinyScript	0/3	0/3	0/4	N.A.	N.A.	N.A.
WASP2	3/3	3/4	2/3	3	9.7	23.5

* Does not support temporal queries.

- WASP and TinySQL are close for T1 and T2, but WASP is easier for T3.
- TinyTemplate, TinyScript: low success rates for archetype.
- Average development time of WASP2 is 47% of WASP.

User study results

Language	Success rate			Develop time (min)		
	T1	T2	T3	T1	T2	T3
SwissQM	3/3	3/3	N.A.*	5.7	11.3	N.A.
WASP	2/2	2/4	2/4	16	31	29.5
TinySQL	3/4	2/3	0/3	17.7	27.5	N.A.
TinyTemplate	1/4	0/3	0/3	34	N.A.	N.A.
TinyScript	0/3	0/3	0/4	N.A.	N.A.	N.A.
WASP2	3/3	3/4	2/3	3	9.7	23.5

* Does not support temporal queries.

- WASP and TinySQL are close for T1 and T2, but WASP is easier for T3.
- TinyTemplate, TinyScript: low success rates for archetype.
- Average development time of WASP2 is 47% of WASP.

Compilation and evaluation on test bed

WASP parser generates fragments of NesC code.

NesC skeleton and library for Archetype I.

Python script combines them and generates complete TinyOS code.

Compile TinyOS code to executables.

Tested 3 tasks with multihop network of TelosB nodes.

Conclusion

To open sensor network to application experts

- Design with novice programmers in mind.
- Evaluate languages with user studies.

Sensor network taxonomy and archetype-specific languages.

WASP programming language for the most frequently encountered archetype.

User study shows novice programmers are more likely to succeed for the most commonly encountered archetype with WASP than other evaluated languages.

Future work

Languages for other archetypes.

Wireless sensor network synthesis from archetype-specific languages.

Project website http://absynth-project.org/.