Construct – Community Middleware for Pervasive Computing

Steve Neely

Systems Research Group
School of Computer Science and Informatics
UCD Dublin Belfield, Dublin 4, Ireland

http://www.csi.ucd.ie/

Core theme

- My research has organically evolved into the core theme:

Making data and information available

(by opening access to services → resources)

A little bit of history

- PhD "*Mobile Computations over Distributed Semistructured Data*" (2003)
  - University of Strathclyde, Glasgow, Scotland
  - Database/programming languages group
  - Built one of the first data binding Java to XML systems
  - I added mobile code queries for network data repositories
- 2005: Systems Research Group at University College Dublin, Ireland (not California or Georgia or Indiana or Maryland or New Hampshire or Pennsylvania or Texas or Virginia or Ontario or even South Australia)
  - UCD CSI UbiComp group
  - Prof. Paddy Nixon, Prof. Simon Dobson, Dr Aaron Quigley, Dr Joe Kiniry, & Dr Chris Bleakley
  - Four postdocs (doing all the work :-) and (supervising) ~30 postgrads

Systems Research Group key themes

- Distributed systems
  - context-awareness, pervasive, ubicomp, autonomics, sensors, web…
- Formal methods
  - specification languages, proofs & correctness, security
- Visualisation
  - presenting large data sets, graph drawing, CAVE…

Commonality: *Pervasive Systems*

(which are hard to build)

Construct goals

- To provide an **open, standards-based platform**
  - to encourage and facilitate the development of pervasive and autonomic systems
- To **simplify** the development of pervasive systems
  - by providing a collection of pre-built **sensors, modules and services**
- To act as a **target for research** into adaptive systems design, and into programming language constructs for such systems
- To nurture a **community of developers** who can build on each others' work

Wait a minute...

- Pervasive computing?
  - aka UbiComp
  - or autonomic computing
  - or ambient intelligence
  - or Internet of things
  - or smart dust
- small, inexpensive, robust networked devices
- Weiser: *weaving of technology (information processing) into the fabric of every day life*
  - user not necessarily aware that the system is there
  - smart fridge
- Peace Corps summary

Pervasive systems - general issues (1)

- May consist of an unbounded number of devices
  - scalability must be addressed
- Decentralization makes construction of pervasive systems a tractable problem
  - allow growth yet retain coordination though lateral relationships between components
  - localized error handling
  - provide better support for extremes of scale
  - reduce # of bottlenecks
  - handle failure more gracefully

Pervasive systems - general issues (2)

- *Ad hoc*-ness makes developers lives miserable
  - device movement
  - takes services and data with it
- proxmity does not imply right of access
  - printer example
- cannot rely on a fixed infrastructure
- very few (if any?) guarantees

Pervasive systems - general issues (3)

- Mirroring services and data
  - redundancy with availability versus resource consumption
- Multi-vendor devices rarely talk to each other
  - standards push is a testament to this
- Human management is infeasible
  - systems are beyond our comprehension
  - self-* is an immediate answer
- And we've not *started* on privacy, security, trust...

Pervasive systems are hard to build

- Pervasive and autonomic systems
  - involve large and dynamically changing populations of components and services
  - are highly adaptive
  - must deal with a variety of sensors delivering partial and uncertain results
- Systems must adapt yet remain stable enough to present a predictable service to users
  - and other systems
- Engineering such systems requires considerable infrastructural development work
  - addressing a wide range of subtle issues

Managing complexity

- Many of these issues are core and infrastructural
  - identify them and:
- Deal with them at a middleware level
  - a common platform on which we can build more advanced applications and services
- Providing plumbing and services to developers
  - separation of concerns; allowing them to concentrate on their problem space
  - focus on optimizing middleware services

Construct is

- …our attempt at a support system
- It is a distributed, fully-decentralized, open-source platform
    - supporting the building of context-aware, adaptive, pervasive and autonomic systems
- Construct handles:
    - data management
    - information processing
    - and ubiquitous entity interactions

and it is

- easily extensible
- comes with a wide-range of ready-to-run
  - sensors
  - applications
  - and other tools for developers of context-aware systems
- component-based for dynamic loading of services and updates
- examples:
  - location tracking
  - weather
  - iCal
  - reviews
  - event guides…

Knowledge-driven systems

- We characterize Construct as *knowledge-driven*
  - applications interact and manipulate a common data model
    - *traditional approach piece systems together*
- Applications are provided with a uniform view of the world
  - even if the underlying structures are in a constant state of flux
- Sensor fusion techniques integrate noisy data sources
  - in a clean, dynamic and semantically well-founded manner

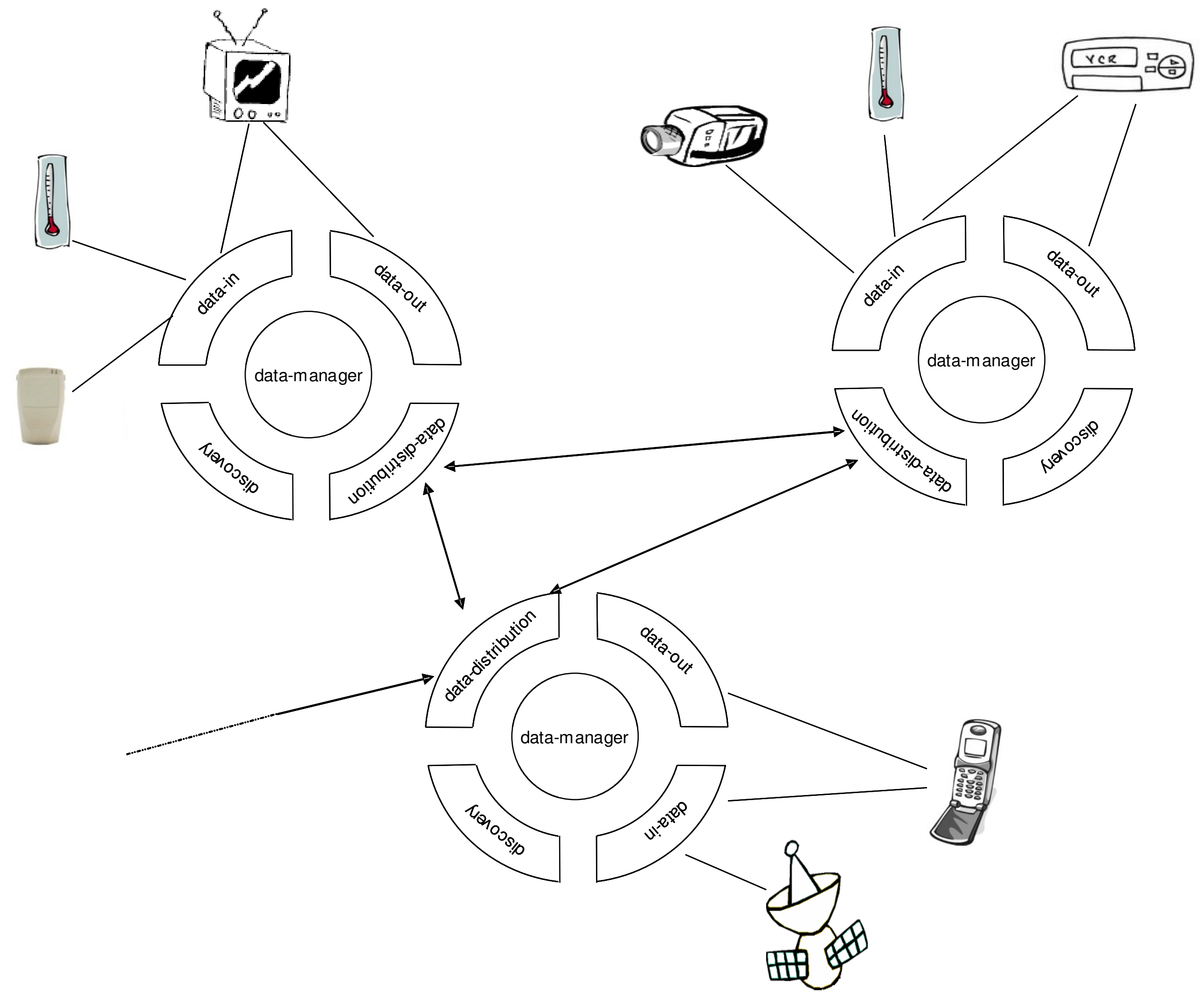
Construct architecture

- Construct nodes distributed as a set of federated peers
  - fully decentralised
  - star topology between individual sensors and their “local” Construct node
- Self-discovery
  - upon start-up peers automatically discover each other and form an overlay
  - and make services available to each other
- Information is *gossiped* throughout the network of nodes
  - fault tolerant – highly redundant

Construct peers

- All peers support (but are not limited to) 5 core services
  - discovery
  - data-manager
  - data-in
  - data-out
  - data-distribution
- All services that interact with an entity are associated with a self-describing manifest
  - details the nature of services and interaction modes
  - manifests are passed to the discovery service on launch

Architecture



Peer roles

- The **data manager** is responsible for validating and storing data on each peer
  - it regulates access that services have to data
- New data enters via the **data-in** service
  - this accepts the data and passes it to the manager
- The **data-out** service is provides a mechanism for querying data in the peer
- Data communication is via the **distribution** service
- The net result is that entities work with a local view of a global data set

Start-up: ZeroConf

- The discovery service
- Bonjour/ZeroConf
- Essentially a style of DNS
  - broadcast to discover local peers
- Implement a set of call-backs and register your services for events
  - this is all handled by our client proxy service for the developer
- Describe your service with a cut-down WSDL
  - find new services by querying their descriptors
- Dynamic rebinding on peer or service failure

Data modelling

- Uses RDF as a common language for representing data
  - N-triple subject, predicate, object
    - *"Steve", "has logical location", "CS Department, Boulder, Colorado"*
  - Wide range of techniques and tools for managing RDF
  - Using the Jena framework
- We store two models
  - entity supplied data
  - house keeping metadata
- Query for it using SPARQL or RDQL

Reasoning

- Sensor fusion algorithms to merge data sources
- Noise and inaccuracies are always expected
  - elections
  - case-based reasoning techniques
  - historical data
  - chaining (forward, reverse)
  - Bayesian
- Inferencing engines for knowledge extraction

Gossiping

- Observation is that sensor data is refreshed
  - so we can relax guarantees of arrival
- Three layers:
  1. data layer exchanges RDF with data managers
  2. gossiping layer drives process
  3. network layer manages physical data packet movement
- Gossiping subsystem
  - core gossiping protocol
    - *description of implementation and evaluating in MPAC paper at Middleware 2006*
  - message buffer
  - peer membership manager
- Summaries gossiped
  - metadata ages data to prevent overwhelming

Metadata

- Used extensively by the system
- Manage sensor data
  - age
  - reliability
  - provenance
- Describing services and applications
  - for discovery, binding and communication
- All extensible and machine interpretable
  - affording self-management

Device representation

- All devices are either sensors or actuators
  - allowing “smart” environments to be monitored and controlled
- These communicate with Construct nodes
- Uniform view of information
  - regardless of how it is derived
- Use Semantic Web technologies
  - to represent and manage data, RDF, Jena, OWL, etc.
- Metadata supporting data management
  - aging of data
  - privacy, ownership

Sensors as information providers

- All information sources are sensors
- Physical sensors
  - e.g. temperature, location awareness
- Virtual sensors
  - aggregation sensors
  - web sensors
- Sensors insert data into their local Construct node

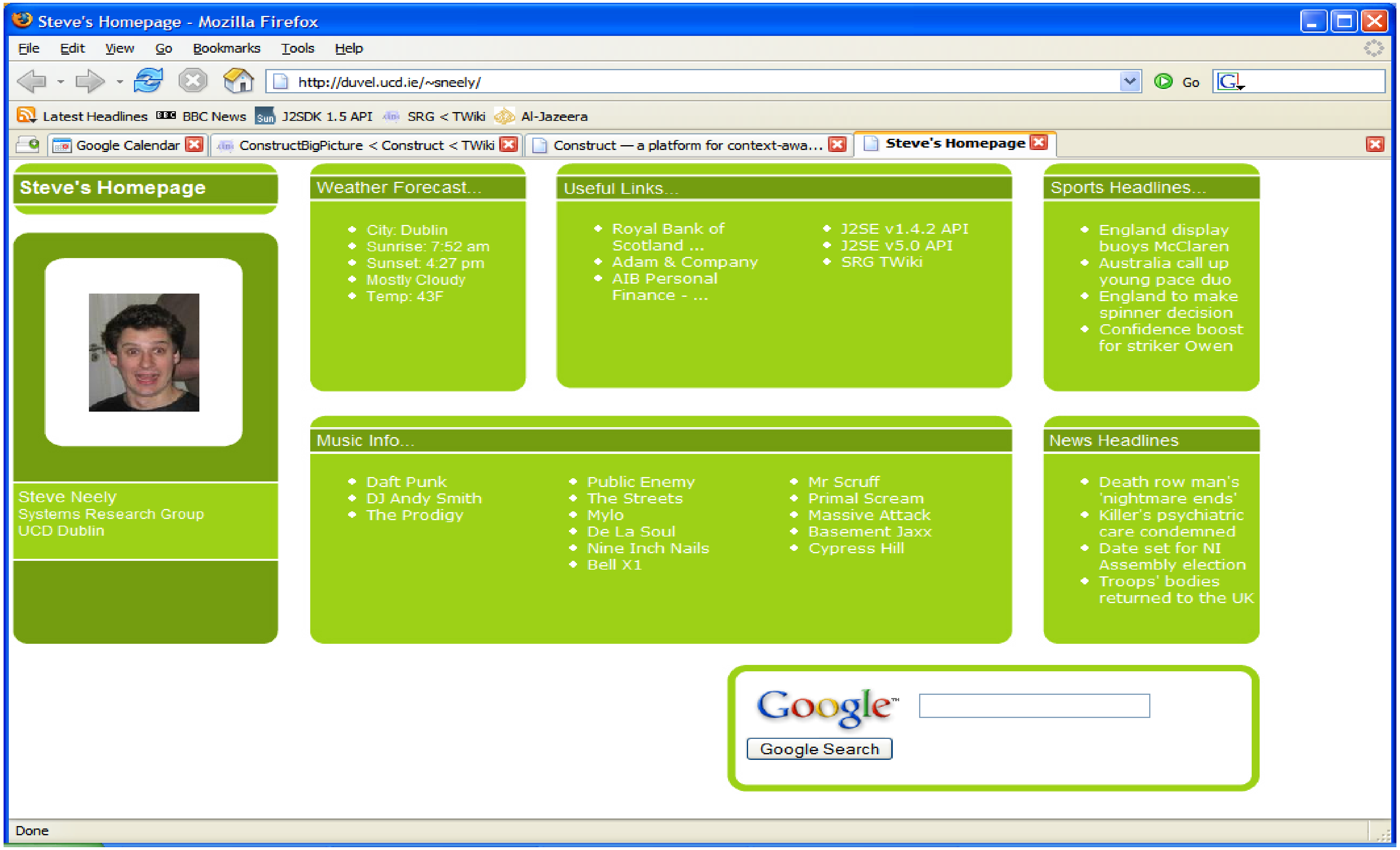
Adding new sensors

- Data are represented using RDF
- Ontologies describe the types of data that exist
  - e.g. new temperature sensors follow the existing ontologies
  - new ontologies are semantically mapped to each other
- Applications query against the ontologies
  - not the sensors
- Applications are decoupled from sensors
  - both logically and physically
  - resource discovery not necessary at an application level
- New/multiple sensors can benefit existing applications

Example: context-personalisation

- Idea: wouldn't it be great to not miss music concerts?
- If we know what a user is listening to we could search the web for gigs for them
- So we did
  - Bunch of Ruby scripts + iTunes library + ticketmaster
- But what if I go to America?
  - Gigs in Dublin are not of interest to me
- Ubisense knows I'm not in CSI
  - and my calendar suggests that I'm in America…

Context-aware web page



Steve's Homepage - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://duvel.ucd.ie/~sneely/

Latest Headlines BBC News J2SDK 1.5 API SRG < TWiki Al-Jazeera

Google Calendar | ConstructBigPicture < Construct < TWiki | Construct — a platform for context-awa... | Steve's Homepage

Steve's Homepage

Steve Neely
Systems Research Group
UCD Dublin

Weather Forecast...

- City: Dublin
- Sunrise: 7:52 am
- Sunset: 4:27 pm
- Mostly Cloudy
- Temp: 43F

Useful Links...

- Royal Bank of Scotland ...
- Adam & Company
- AIB Personal Finance - ...
- J2SE v1.4.2 API
- J2SE v5.0 API
- SRG TWiki

Sports Headlines...

- England display buoys McClaren
- Australia call up young pace duo
- England to make spinner decision
- Confidence boost for striker Owen

Music Info...

- Daft Punk
- DJ Andy Smith
- The Prodigy
- Public Enemy
- The Streets
- Mylo
- De La Soul
- Nine Inch Nails
- Bell X1
- Mr Scruff
- Primal Scream
- Massive Attack
- Basement Jaxx
- Cypress Hill

News Headlines

- Death row man's 'nightmare ends'
- Killer's psychiatric care condemned
- Date set for NI Assembly election
- Troops' bodies returned to the UK

Google Search

Done

Construct enabled webpage

- But I’m collecting all data on my own
    - if I write a new Ruby script to query American concert listings I could share my findings
- Solution:
- Build sensors (real or virtual) and they drop data into Construct
- We write applications against the ontology for that data
    - protection from different underlying representations
- New sensors add to the richness of the data soup
    - applications suddenly become more accurate without being rewritten

Construct key features (1)

- Construct differs from other pervasive systems platforms in a number of key respects:
- It is completely standards-based
  - using RDF as its data exchange model
  - ZeroConf for resource discovery
- It supports a knowledge-centric model of interaction
  - where clients' actions are driven by queries and events regarding context of the system

Construct key features(2)

- It uses gossiping to maintain a consistent state across a distributed data structure
  - maximises robustness
  - and scalability
  - and avoids many problems with hot-spots or overloaded paths in communications
- it treats all information sources uniformly as sensors
  - acting as inputs to uncertain reasoning algorithms

Summary (1)

- Construct is a community middleware for context-aware systems
  - provides a core set of services required by these systems
- Automatic data management, distribution and processing is handled by Construct
  - developers do not need to be concerned with common issues of scale and propagation
- Discovery of services and data is automatically dealt with by Construct
  - developers do not need to employ their own complex registry/discovery services
  - dynamic rebinding on failure
    - *where possible*

Summary (2)

- The component-based architecture of Construct is carefully designed to be easily extensible
  - developers can rapidly deploy their own sensors, services and applications for experimentation
- Future releases:
  - more sophisticated reasoning and knowledge extraction through inferencing engines
  - provenance data for tracking lineage
  - programming language bindings to core ontology structures

Conclusion – what did we learn?

- Pervasive systems are *still* difficult to build
  - they are complicated to understand and visualize
- middleware like Construct *can* ease development of context-aware applications
  - it takes care of the plumbing
- there is a learning curve to address
  - must understand the ontology descriptors & SW technologies
- and the ground is always changing
  - partly due to our approach of adopting a wide variety of relatively young technologies

References & acknowledgements

Me: steve@steveneely.org

Construct: http://www.construct-infrastructure.org/

Thanks to Science Foundation Ireland under which this work was supported on grant 04/RPI/1544