1. Record Nr. UNINA9910811310703321

Autore	Bakshi Amol B. <1975->
Titolo	Architecture-independent programming for wireless sensor networks / / Amol B. Bakshi, Viktor K. Prasanna
Pubbl/distr/stampa	Hoboken, N.J., : J. Wiley-Interscience, c2008
ISBN	9786611381585 9781281381583 1281381586 9780470289303 0470289309 9780470289297 0470289295
Edizione	[1st ed.]
Descrizione fisica	1 online resource (209 p.)
Collana	Wiley series on parallel and distributed computing
Altri autori (Persone)	Prasanna KumarV. K
Disciplina	681/.2
Soggetti	Sensor networks - Programming Wireless LANs - Programming
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Note generali	Description based upon print version of record.
Nota di bibliografia	Includes bibliographical references (p. 179-183) and index.
Nota di contenuto	Preface -- Acknowledgments -- 1. Introduction -- 1.1 Sensor networks and traditional distributed systems -- 1.2 Programming of distributed sensor networks -- 1.2.1 Layers of programming abstraction -- 1.2.1.1 Service-oriented specification -- 1.2.1.2 Macroprogramming -- 1.2.1.3 Node-centric programming -- 1.2.2 Lessons from parallel and distributed computing -- 1.3 Macroprogramming: What and Why? -- 1.4 Contributions and Outline -- 2. The Abstract Task Graph -- 2.1 Target applications and architectures -- 2.2 Key Concepts -- 2.2.1 Data Driven Computing -- 2.2.1.1 Program flow mechanisms -- 2.2.1.2 Why data driven? -- 2.2.2 Mixed Imperative-Declarative Specification -- 2.3 Syntax -- 2.3.1 The Structure of an ATaG Program -- 2.3.2 More on Task Annotations -- 2.3.3 Illustrative examples -- 2.4 Semantics -- 2.4.1 Terminology -- 2.4.2 Firing rules -- 2.4.3 Task graph execution -- 2.4.4 get() and put() -- 2.5 Programming idioms -- 2.5.1 Object tracking -- 2.5.2 Interaction within local neighborhoods -- 2.5.3 In-network aggregation -- 2.5.4 Hierarchical data fusion --

2.5.5 Event-triggered behavior instantiation -- 2.6 Future work -- 2.6.1 State-based dynamic behaviors -- 2.6.2 Resource management in the runtime system -- 2.6.3 Utility based negotiation for task scheduling and resource allocation -- 2.6.4 Analyzing feasibility of compilation -- 3. DART:The Data Driven ATaG Runtime -- 3.1 Design objectives -- 3.1.1 Support for ATaG semantics -- 3.1.2 Platform independence -- 3.1.3 Component-based design -- 3.1.4 Ease of software synthesis -- 3.2 Overview -- 3.3 Components and functionalities -- 3.3.1 Task, Data, and Channel Declarations -- 3.3.2 UserTask -- 3.3.2.1 Service -- 3.3.2.2 Interactions -- 3.3.2.3 Implementation -- 3.3.3 DataPool -- 3.3.3.1 Service -- 3.3.3.2 Interactions -- 3.3.3.3 Implementation -- 3.3.4 AtagManager -- 3.3.4.1 Service -- 3.3.4.2 Interactions -- 3.3.4.3 Implementation -- 3.3.5 NetworkStack -- 3.3.5.1 Service -- 3.3.5.2 Interactions -- 3.3.5.3 Implementation.
3.3.6 NetworkArchitecture -- 3.3.6.1 Service -- 3.3.6.2 Interactions -- 3.3.6.3 Implementation -- 3.3.7 Dispatcher -- 3.3.7.1 Service -- 3.3.7.2 Interactions -- 3.3.7.3 Implementation -- 3.4 Control flow -- 3.4.1 Startup -- 3.4.2 get() and put() -- 3.4.3 Illustrative example -- 3.5 Future work -- 3.5.1 Lazy compilation of channel annotations -- 3.5.2 Automatic priority assignment for task scheduling -- 4. Programming and Software Synthesis -- 4.1 Terminology -- 4.2 Meta-modeling for the ATaG domain -- 4.2.1 Objectives -- 4.2.2 Application model -- 4.2.3 Network model -- 4.3 The programming interface -- 4.4 Compilation and software synthesis -- 4.4.1 Translating task annotations -- 4.4.2 Automatic software synthesis -- 4.4.3 The ATaG simulator -- 4.4.4 Initialization -- 4.4.4.1 Situatedness -- 4.4.4.2 Network interface -- 4.4.4.3 Network architecture -- 4.4.4.4 Sensor interface -- 4.4.5 Visualizing synthesized application behavior -- 5 Case Study: Application Development with ATaG -- 5.1 Overview of the use case -- 5.2 Designing the macroprograms -- 5.2.1 Temperature gradient monitoring -- 5.2.2 Object detection and tracking -- 5.3 Specifying the declarative portion -- 5.4 Imperative portion: Temperature gradient monitoring -- 5.4.1 Abstract data items: Temperature and Fire -- 5.4.2 Abstract Task: Monitor -- 5.4.3 Abstract Task: Temperature Sampler -- 5.4.4 Abstract Task: Alarm Actuator -- 5.5 Imperative portion: Object detection and tracking -- 5.5.1 Abstract data items: TargetAlert and TargetInfo -- 5.5.2 Abstract Task: SampleAndThreshold -- 5.5.3 Abstract Task: Leader -- 5.5.4 Abstract Task: Supervisor -- 5.6 Application Composition -- 5.7 Software Synthesis -- 6 Concluding Remarks -- 6.1 A framework for domain-specific application development -- 6.2 A framework for compilation and software synthesis -- References.

Sommario/riassunto

New automated, application-independent methodology for designing and deploying sensor networks Following this book's clear explanations, examples, and illustrations, domain experts can design and deploy nontrivial networked sensing applications without much knowledge of the low-level networking aspects of deployment. This new approach is based on the Abstract Task Graph (ATaG), a data-driven programming model and an innovative methodology forarchitecture-independent programming and automatic software synthesis for sensor networks. ATaG programs are concise, understandable, and network-independent descriptions of global application functionality that can be automatically compiled onto any target deployment. The book begins with an overview chapter that addresses the important issues of programming methodologies and compares various programming models for sensor networks. Next, the authors set forth everything you need for designing and deploying

sensor networks using ATaG, including: . Detailed description of the ATaG model's features . System-level support for architecture-independent programming . Examination of the graphical programming and software synthesis environment for ATaG . Case study illustrating the process of end-to-end application development and software synthesis using ATaG Throughout the book, the authors provide code excerpts and figures to help clarify key concepts and explain each step. For programmers, the graphical formalism of the ATaG program, coupled with the fact it uses an existing language (Java), means that no special training is needed to start developing and deploying applications in ATaG. Everything you need to know is clearly set forth in this book.