多段階モデル駆動開発を用いた
無線センサネットワークソフトウェア
開発手法に関する研究

Study on Multi-Level Model-Driven Development Approach for Wireless Sensor Network Software

2015 年 2 月

早稲田大学大学院 基幹理工学研究科
情報理工学専攻 ソフトウェア開発工学研究

清　水　　遼
Ryo SHIMIZU

目 次

図 目 次

表 目 次

第1章　序論

1.1　無線センサネットワーク

コンピュータ機器の小型化や低価格化に伴い，実世界の様々なものに埋め込まれ，実世界と連動するシステムの実用化が産学双方にて取り組まれている．このようなシステムはCyber-Physical Systems (CPS) [105] や Internet of Things (IoT) [56] などと呼ばれ注目されており，センサ機器を通して得た実世界の情報に基づき自身の置かれた状況を分析し，振る舞いを決定，アクチュエータ機器を通して実世界に影響を与える．例えばトンネル監視システム [17] では，センサを用いてトンネル内の輝度を計測し，データを分析，アクチュエータを通しトンネル内の照明の明るさを管理している．この様に観測，分析，実行を自動化するで，人手での管理コストの低減や，人手では困難な利用資源の最適化が可能である．実世界の情報を取得する手段として，センサの計測値を，複数のデバイス (センサノード) 間の無線通信により伝播，収集する無線センサネットワーク（WSN: Wireless Sensor Network）が用いられる [56, 105]．

WSN の特徴として，WSN を構成するセンサノードが小型のデバイスで無線通信機能を持つこと，バッテリ駆動であること等が挙げられる [107]．また，無線通信技術や電子機器の製造技術の進歩により従来のセンサ機器よりも低価格であること，通信や電源供給に有線を用いないため敷設時の制限が少ないといった特徴も有する [4]．更には，センサノードは小型ゆえに限られた計算資源しか持たないものの，用途に応じてソフトウェアを配備可能なプログラマブルデバイスである．このようなセンサノードやその上で動作する OS は成熟期を迎えており，OS 上で動作するソフトウェアを開発，配備することで用途を変更することが可能となっている．

これらの特徴から WSN は，様々な分野への適用とその需要増加が見込まれている．WSN は従来センサ機器が設置されてきた場所に加え，広域で人手が届きにくい場所やデバイスやケーブルの敷設，電源の確保が困難な場所，美観などの問題でケーブル等の設置に制限のある既存の建築物へと適用範囲を広げており，周辺環境の監視 [18]，スマートビルディング [2]，医療分野 [21] など種々の分野に対する適用が試みられている．IDTechEX 社の調査によれば，WSN の市場規模は 2013 年時点では約 4.5 億米ドルであるが，2022 年には

約 20 億米ドルと約 4 倍になるとされており，その有用性からも WSN の市場規模は今後より一層の普及が見込まれている [39].

1.2　無線センサネットワークソフトウェア開発の現状と問題

WSN は主たる機能要求であるデータ処理と同時に非機能要求を満たすよう開発される．特に，より正確に，長期間データ処理を継続するためには，WSN では取得したデータに関する品質 (以降，単に品質と呼ぶ) が重要な非機能要求であることが知られている [19, 82]．このような品質は WSN の配備方法や障害物といったソフトウェアが実行される環境に強く影響を受けること [92, 96] や，非機能要求は WSN の適用先ごとに異なること [66] が知られており，対象とする環境ごとに非機能要求を満たすよう品質を改善すること，すなわちシステムを設計することが求められる [84, 52].

WSN における品質改善では WSN ソフトウェアの設計が重要となる．WSN システムはソフトウェア，OS，ハードウェア，ノード配備に関わる種々の設計により構成され，これらの設計の決定や変更により品質を改善する．しかしながら，要求を満たすための設計には制約やコストの問題が併存する．例えば，OS は既に成熟したものを用いる場合がほとんどであり変更が難しい．また変更する場合，その設計はノード動作の根幹を担うため注意深く決定，変更する必要がある．或いは，ハードウェアやノード配置の設計変更には，一度配備したセンサノードの変更・移動を要するため人手による手間がかかる [99]．一方でソフトウェアの設計は OS 上での動作のみを決定，変更すること，ネットワークを介してプログラムの再配備が可能なことから，前述の設計に比べ容易に決定や変更を実施可能である．すなわちソフトウェアの設計は変更容易性が高く，品質改善において高い重要性を持つ.

このような WSN ソフトウェアは，通常のソフトウェアと同様，新規開発，移植開発の 2 通りで開発されうる．既存成果物を移植することは，全体或いは部分的にソフトウェアを再利用することにつながり開発コストを低減可能であるために望ましい．また，共有センサネットワーク [30, 93] のような WSN では，センサノードが既に敷設済みであり，OS やミドルウェアも WSN の管理者により決定されるため，対象とする環境により OS やミドルウェアが異なることが起こりうる．或いは，サポート切れなどにより既存の OS やミドルウェアが陳腐化する場合も起こりうる [87]．このような場合，既存のソフトウェアを異なるプラットフォームへ移植する必要性が生じうる．しかしながら，WSN ソフトウェアへの要求は適用先により多様であり [62, 66]，必ずしも移植可能な成果物が存在するとは限らない．この場合，要求に対応する WSN ソフトウェアを新規に開発する必要が生じる．

また WSN ソフトウェア開発は，開発ごとに選択されるプラットフォームを対象に行われる．WSN ソフトウェア開発支援のため，これまで数多くのドメイン特化言語 (DSL: Domain-Specific Language) が，その実行環境である OS やミドルウェアと合わせてソフトウェアプラットフォームとして提案されてきた．WSN ソフトウェアでは主に，個々のセンサノードに対する処理を分散プログラムとして記述する視点を開発者に与えるプラットフォームが用いられる．以降，本論文では開発者与える視点を抽象度と呼び，個々のノードに対する視点を Node-level，Node-level の抽象度を持つプラットフォームを Node-level プラットフォームとそれぞれ呼ぶ．Node-level プラットフォーム上では，開発者は主たる機能要求であるデータ処理の設計，実装と同時に，主たる非機能要求であるデータに関する品質，特に通信やタスク割当の設計，実装も扱う．Node-level プラットフォームではその開発は複雑となるものの，通信やタスク割当まで含めた細やかな設計変更による品質改善が可能である．或いは，個々のセンサノードにまたがる分散処理設計を捨象しノード集合の処理の視点 (Group-level) や，WSN 内の通信設計までを捨象することで WSN 全体の処理視点 (Dataflow-level) を提供し，プログラムの記述を簡潔にするプラットフォームも提案されている．この様なプラットフォーム上では，捨象された設計は DSL では変更不可能であり，実行ミドルウェア側が持つ固定的な設計で実行され品質改善が難しくなる．WSN ソフトウェア開発では品質改善が非常に重要視されるが，これらのプラットフォームでは品質改善のための設計変更が支援されていない．したがって本研究では，柔軟な設計変更可能な Node-level プラットフォームを対象とする．

更に，WSN ソフトウェア開発は，開発対象ごとに品質を改善するよう実施される．品質改善は，まず初期実装を獲得し，その実行により品質測定と設計課題の特定を行い，適切な設計変更を実施することで達成される．その後は設計変更を施した実装を再度実行，品質を測定，課題の特定と設計変更を繰り返すことで段階的な品質改善を試みる．この際，初期実装を獲得する工程においては，開発の早期段階で設計上の課題を特定するためには，低コストで初期実装を得られることが望ましい．すなわち，(要件 1.) 新規開発において，低コストでプロトタイプ開発を開発可能なこと，(要件 2.) 移植開発において，既存成果物を低コストで移植可能なこと，が求められる．また品質改善を試みる工程においては，品質と生産性を高めるため，(要件 3.) 通信，タスク割当設計を含めた細やかな設計変更が可能なこと，(要件 4.) 品質改善のための設計変更を端的に記述可能なこと，が求められる．・このような WSN ソフトウェア開発を支援するため，現状では，ソフトウェア工学分野の知見であるモデル駆動開発 (MDD: Model-Driven Development) の適用が試みられている [57]．MDD では，開発者は実装の抽象表現であるモデルを成果物の中心に据え，変換による詳細化を通して実装獲得を試みる開発手法である [81]．既存の多くの MDD 手法は Node-level プラットフォームを対象に，モデルによる抽象表現，モデルからのコード生成による開発

支援を提案している．また WSN ソフトウェアの移植開発に向けては，MDD の特種であるモデル駆動アーキテクチャ(MDA: Model-Driven Architecture) [64] を用いる手法が提案されている．MDA では，プラットフォーム非依存なモデル (PIM: Platform Independent Model) を、変換を通して複数のプラットフォーム特化なモデル (PSM: Platform Specific Model) へと詳細化することでプラットフォーム間移植を実現する．既存の MDD の手法は，サーベイ論文により，MDD 手法が適用するモデリング言語を基準に，3 段階の抽象度 (Node-, Group-, Dataflow-level) による分類が可能なことが示されている．

しかしながら，現状の開発手法では開発対象ごとの品質改善に求められる 4 つの要件を同時に達成することが困難である．Dataflow-level や Group-level に属する MDD 手法を用いる場合，通信やタスク割当の設計がモデリング言語上で対象外となっており，“要件 3. 通信，タスク割当設計を含めた細やかな設計変更が可能なこと” を達成できない．また，Node-level に属する MDD を用いる場合，モデリング言語上でデータ処理設計と同時に通信設計，タスク割当設計も扱わなければならず記述が複雑化するため，“要件 1. 新規開発において，低コストでプロトタイプ開発を開発可能なこと”，“要件 4. 品質改善のための設計変更を端的に記述可能なこと” の双方を達成できない．更に，いずれの手法も単一の抽象度のみを用いており，多様なプラットフォームが存在する WSN 分野では移植元 PSM と PIM が持つ情報量に乖離が生じる．この乖離は開発者が手動で埋めなければならないため，“要件 2. 移植開発において，既存成果物を低コストで移植可能なこと” を達成できない．すなわち，現状の開発手法では，開発対象ごとの品質改善を実現するための 4 つの要件を同時に達成できない．

1.3　本論文の目的

本論文は，前節で述べた，品質改善のための要件を満たせいないという問題の解決を目指す．新規開発，移植開発それぞれにおいてこの問題を解決するための共通基盤として，複数のドメイン特化モデリング言語 (DSML: Domain-Specific Modeling Language) を併用し，記述されたモデルを段階的に詳細化可能なモデル駆動開発（MDD: Model-Driven Development）フレームワークを提案する．モデル駆動開発とは，実装よりも抽象度の高いモデルをソフトウェア開発における成果物の中心に据え，モデル変換，特に自動化された変換やコード生成を通して段階的にモデルを実装へと近づけていく開発手法である [81, 41]．モデルを用いた抽象化により，問題領域と解決領域との乖離を埋めることでソフトウェア開発の品質及び生産性の向上を図っている [33]．

提案する MDD フレームワークは，3 つの DSML と，DSML で記述されたモデル間を

つなぐモデル変換で構成される．3 つの DSML，DataflowML，GroupML，NodeML は，分散ソフトウェア開発におけるデータフロー設計，タスクアーキテクチャ設計，詳細設計の 3 つの工程それぞれに対応するよう定義した．また，各 DSML の定義は既存の WSN 向けプラットフォームの分類 [62] にも基づき行った．この分類は，数あるプラットフォームを，それらが開発者に提供する抽象化の視点から 3 つの抽象度に分類している．最も抽象度の高い DataflowML は，WSN ソフトウェアが達成すべきデータ処理を，ネットワークに非依存なデータフロー設計として記述するための言語として定義した．また中間の抽象度に対応する GroupML は，WSN ソフトウェアが達成すべきデータフローを，ノードグループの構成とグループ間の通信設計として記述するための言語として定義した．最後に，最も抽象度の低い NodeML は，分散処理設計を個々のノードが達成すべきタスクとその割当として記述するための言語として定義した．これら 3 つの DSML の併用により，開発者に複数の抽象化の視点や段階的な詳細化を提供する．

しかしながら，複数の DSML を併用する場合，全てのモデルをスクラッチから記述すると，記述内容の重複によりモデルの記述コストが増加しうる．また，人手でのモデル記述や修正により，異なる DSML で記述されたモデル間で，モデルが持つ情報に齟齬が生じうる．これに対し本論文では，抽象度の高いモデルから低いモデルへと情報を伝播させるモデル変換規則を定義することで，自動生成によりモデル記述コストを低減しつつ，3 種のモデル間で単方向の一貫性を確保した．

新規開発に対しては，提案フレームワークを用いた，3 つの DSML を開発工程ごとに使い分ける開発プロセスを提案する．提案プロセスはプロトタイプ開発工程と品質改善工程の 2 つに分けられる．プロトタイプ開発工程では，開発者はデータフロー設計のみを行い，DataflowML を用いてこれを記述する．その後提案フレームワークは，通信方式の設計を反映する GroupML により記述されたモデル，タスク割当の設計を反映する NodeML により記述されたモデル，実行可能な実装コードを自動生成により出力する．すなわち，本工程ではデータフロー設計のみを扱うことで低コストでのプロトタイプ開発を達成する．品質改善工程では，前工程で記述及び自動生成されたモデルから任意の抽象度を選択し設計変更を行うことで品質改善を試みる．実施された設計変更はモデル変換，コード生成を通して実装コードまで反映される．すなわち，本工程ではデータフロー，通信方式，タスク割当の全ての設計を対象とすることで Node-level プラットフォームにおける細やかでかつ端的な設計変更も達成する．この様に各工程にて DSML を使い分けることで，Node-level プラットフォームを対象とした新規開発において，低コストでのプロトタイプ開発と，端的で細やかな設計変更を達成する開発プロセスの実現を目指す．

移植開発に対しては，提案フレームワークを利用した，複数プラットフォーム対応のソフトウェア移植手法を提案する．提案する開発手法では，提案した 3 つの抽象度の DSML

で記述されたモデルを PIM とみなす．これらの PIM は WSN 向けプラットフォームの分類をもとに定義されており，移植時には移植元 PSM を，移植元プラットフォームが属する抽象度に対応する PIM に向けて変換する．これにより移植元 PSM に含まれる情報を引き継いだ PIM を生成し，本フレームワークの提供する変換規則によりコード生成に必要な情報を補完する．多段階 PIM と PIM 間の変換規則により人手でのコストを低減することで，低コストでの移植の実現を目指す．移植による実装獲得後の品質改善工程は，新規開発の場合と同様である．

1.4　本論文の構成

本論文の構成は以下の通りである．まず第 2 章では，本章で述べた WSN ソフトウェア開発とその問題について，具体例を用いて詳述する．続く第 3 章では，第 2 章で述べた問題を解決するために用いる MDD レームワークを提案し，その構成要素となるモデリング言語とモデル変換規則について詳述する．また提案フレームワークを実現する実装詳細についても述べる．第 4 章では，提案フレームワークを用いた新規開発のための開発プロセスを提案し，その評価と有用性についても論じる．第 5 章では，提案フレームワークを用いた，移植開発における複数プラットフォーム対応のため開発手法を提案する第 6 章では，提案手法，評価の結果を受け，本研究の限界や有効範囲についての考察を詳述する．更に，第 7 章では WSN 分野，MDD 分野の双方における本研究の位置づけを，関連研究を示しつつ論じる．最後に第 8 章にて本論文で提案手法の総括を行い，更に今後の課題について整理し本論文をまとめる．

第2章　無線センサネットワークソフトウェア開発と品質改善の支援

本章では，現状の WSN ソフトウェア開発における問題を詳述する．まず 2.1 節では本論文で対象とする WSN ソフトウェアの種類を明示し，以降で用いる WSN ソフトウェア例を挙げる．続く 2.2 節では，WSN ソフトウェアの品質改善と，品質改善時に求められる要件について述べる．2.3 節，2.4 節では既存の WSN ソフトウェア開発支援技術について詳述する．2.3 節では既存の WSN 向けソフトウェアプラットフォームとその得失について説明し，本論文が対象とするプラットフォームの種類を明示する．2.4 節では最新の研究動向として MDD を WSN ソフトウェア開発に適用した研究について述べる．最後に 2.5 節にて既存の WSN 向け MDD の手法の限界と本研究が取り組むべき問題について詳述する．

2.1　対象とする WSN ソフトウェア

WSN ソフトウェアは，WSN の適用分野の多様さから様々な種類が存在し，幾つかの既存研究においてその分類がなされている．Bai らはノードの移動性や計測・送信の実行タイミングなど 8 つの視点から 23 のソフトウェアに対し分類を試みている [8]．また，Mottola らは，WSN ソフトウェアの達成すべき目的やデータの計測タイミングを含む 5 つの観点から 28 の WSN ソフトウェアを分類している [62]．更に Oppermann らは，11 の観点から 62 もの WSN ソフトウェアの分類を試みている [66]．表 2.1 は，上記の 3 つの分類のうち，2 つ以上の分類にて用いられている観点を抽出し，各分類に最も多くの WSN ソフトウェアに該当する属性をまとめたものである．括弧内の数字は該当するソフトウェア数を示している．Goal の観点は WSN ソフトウェアの目的を意味し，計測のみ (SO: Sense-Only) がいずれにおいても最も多くの WSN ソフトウェアに該当した．Mobility の観点は WSN ソフトウェアが移動性を持つノードを対象とするかを意味し，静的なノードを対象とする (Stationary or Static) 場合がいずれにおいても最多であった．Sampling time は WSN ソフトウェアにおけるデータ計測の実行タイミングを示しており，定期的なデータ計測（とデータ送信）を行う WSN ソフトウェアが最も多い．

表 2.1: WSN ソフトウェアの分類のまとめ

調査文献	調査数	**Goal**	**Mobility**	**Sampling time**
Bai らの分類 [8]	23	*Not applied*	Stationary (15)	Periodic (19)
Mottola らの分類 [62]	28	Sense-only (21)	Static (23)	Periodic (20)
Oppermann らの分類 [66]	62	Sense-only (58)	Static (45)	Periodic (47)

表 2.1 より，最も一般的な WSN ソフトウェアは，計測のみを目的とし，移動性を考慮せず，データの計測・送信を定期的に行うものであることがわかる．したがって本論文はこの様な最も一般的な WSN ソフトウェアを開発対象とする．ただし移動性については，実行対象となる配備したセンタノードが移動性を持つ場合であっても，WSN ソフトウェアのロジック上で直接移動性を扱わないものは開発対象に含める．

以降ではこのような種類の WSN ソフトウェアの具体例として，歴史的建造物の監視 (HBM: Heritage Building Monitor) [18] を用いる．この WSN ソフトウェアはイタリアにある歴史的建造物の状態異常監視及び検知を目的としており，16 個のセンサノードと 1 個のベースステーションを配備し，4ヶ月間に渡り運用された実績を持つ．HBM は，配備されたセンサノードが加速度，歪み，温度，湿度の 4 種類のデータを定期的に計測・送信し，集められたデータをもとに建築物に異常がないかを調べるというものである．ただし，歪みのデータについては，計測値が変動しやすいため過去 10 回の計測値の平均値を異常検知に用いている．WSN ではデータに対する集約・融合処理をネットワーク内で実行することも可能であるが，HBM では建築物の異常検知処理はネットワークの外で実行し，歪みデータの平均値計算は個々のノード上で実行している．この様に HBM では，静的な環境の監視を，定期的なデータ計測と送信及びデータ処理により実現している．

2.2　WSN ソフトウェアの品質改善と要件

前節で述べたような WSN ソフトウェアの開発においては，開発対象ごとに品質を高めることが重要であることは 1.2 節でも述べた．本研究では WSN ソフトウェアにおける品質の中でも，特に取得したデータに関する品質を扱う．このような品質の例としては，データ損失率，データ計測期間，データ鮮度、データ精度が挙げられる．データ損失率とは通信時に損なわれたデータの割合を指す．データ計測期間とは，対象データを計測可能な時間の長さを指す．データ鮮度とは得られたデータの新しさを，データ精度とは得られたデータが物理世界を表す正確さをそれぞれ指す．これらの中でもデータ損失率とデータ計測期間は WSN において特に重要な品質特性であることが知られている [16]．

これらの品質は，データフロー，通信方式，タスク割当といった，WSN ソフトウェアに必要な設計上の決定を変更することで改善可能である．WSN ソフトウェアは，機能性要求に相当する，どのような種類のデータが必要か，どのようなデータ集約／融合処理が必要かを表現するデータフローを，複数のセンサノードで構成される分散ネットワーク上で実現する．すなわち，開発者は設計段階において (i) データフロー設計，(ii) データフローの複数タスクへの分割とタスク間の通信方式決定からなるタスクアーキテクチャ設計，(iii) 分割したタスクのノードへの割当からなる詳細設計の 3 つを行う必要がある．特に品質改善には通信設計，タスク割当設計が重要であり，例えば通信設計においてルーティングプロトコルを変更すること，タスク割当設計においてタスクを割り当てるノード数を変更することが品質に影響を与えることが報告されている [96, 100]．

このように前述の設計はデータに関する品質と関連しており，設計を変更することは品質の変化につながる．例えば，前節で述べた HBM では，品質改善のための設計として通信時のデータ圧縮 (=通信方式設計) とノード位置に基づくタスク割当 (=タスク割当設計) が採用されている．HBM では加速度が高頻度で計測されるため，加速度データの計測と送信を担うノードでは通信量が増加し，データ計測期間が短くなるという品質低下を招く．この品質改善のため，Ceriotti らは加速度データのみ圧縮して送受信を行う通信方式を採用している．また，センサノードの配置場所，すなわちデータの計測箇所は，建築学の知識に基づき異常の兆候を早期に検知できるよう選択されている．つまり，タスク割当をノードの物理的な配置場所によって決定することで，最終的に得られる異常判定のデータ精度の向上を計っている．したがって HBM では，配備されたすべてのセンサノードに対して同じタスクを割り当てず，センサノードの配置場所に基づき，ノードごとに扱うデータの種類が異なるようタスク割当が行われている．この様に WSN ソフトウェアの設計を注意深く決定，変更することで品質の改善が可能となる．表 2.2 は，前述の品質と，各品質を改善するための設計例をまとめている．

WSN ソフトウェア開発における品質改善は，反復型のプロセスに従い実施されることが望ましい．図 2.1 に，WSN ソフトウェアの品質改善プロセスの概要を示す．WSN ソフトウェアの品質改善は，実行，監視，分析，修正の繰り返しを主軸に実現される．また，品質改善は主に実行可能な初期実装を得る工程と，品質改善を実施する工程に大別できる．

実行可能な初期実装を得る工程では，主に機能要求を満たすようなソフトウェアの設計，実装を行う．新規開発においては，これはスクラッチからの設計と実装によるプロトタイプ開発に相当する．移植開発においては，これは既存のソフトウェア成果物を，対象とするプラットフォーム上で実行可能な実装へと変換することに相当する．

品質改善を行う工程では，まず前工程にて得られた実装を，対象とする環境にて実行し，実行結果を監視することで品質を取得する．続いて，実行結果から非機能要求を満た

表 2.2: データに関する品質と品質改善のための設計例

品質	データフロー	通信方式	タスク割当
データ損失率	なし	ルーティングプロトコル変更	タスク割当ノード数の増加
データ計測期間	計測間隔や通信間隔の伸長	データ圧縮の採用	タスク割当ノード数の減少
データ鮮度	計測間隔や通信間隔の短縮	ルーティングプロトコルの変更	なし
データ精度	計測間隔や通信間隔の短縮	なし	タスク割当ノード数の増加

す上での設計課題を分析，同定し，具体的な設計変更を決定する．その後，決定された設計変更を実際のソフトウェア成果物へと反映させるための修正を実施する．更には修正された成果物に対応する実装を実行，監視，分析，修正することで，繰り返しによる品質改善を試みる．

前者の工程においては，低コストで初期実装が獲得可能であることが望まれる．品質改善では反復型のプロセスにより試行と修正を繰り返す．この試行錯誤を行う工程に注力するためにも，開発の早期段階にて設計上の課題を同定することは重要である．すなわち，開発の早期に初期実装を獲得することが望ましく，低コストで初期実装が得られることが求められる．

また後者の工程では，開発対象ごとの品質改善を達成するため，細やかな設計変更が端的な表現で可能であることが望まれる．WSN ソフトウェアの設計と実装を変更することは品質改善と直結している．この際，データ処理に関する設計だけを変更する場合と比べ，通信やタスク割当の設計を含めた細やかな設計変更が可能な場合の方が，より品質の良い WSN ソフトウェアを獲得能になると考えられる．また，WSN は分散システムであり，関心事の混在や分散により設計，実装の記述が冗長になりがちであるが，設計変更を端的に記述可能であれば，少ない記述コストで品質を改善可能になると考えられる．従って品質改善では通信，タスク割当を含む細やかな設計変更が可能であり，かつ設計変更が端的に記述可能であることが求められる．

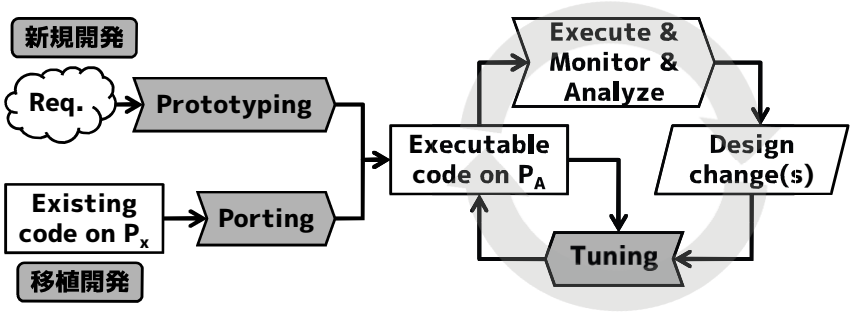


図 2.1: 反復型プロセスによる WSN ソフトウェアの品質改善

以上をまとめると，開発対象ごとに品質改善を達成するためには以下の 4 つの要件を満たす必要がある．

- 初期実装獲得工程における要件

  要件 1. 新規開発における低コストでのプロトタイプ開発が可能であること

  要件 2. 移植開発における多様な WSN プラットフォームからの低コストでの移植が可能であること

- 品質改善工程における要件

  要件 3. 通信設計，タスク割当設計を含む細やかな設計変更が可能であること

  要件 4. 品質改善のための設計変更を端的に記述可能であること

2.3 WSN ソフトウェアプラットフォーム

2.3.1 WSN ソフトウェアプラットフォームとその分類

現状，2.1 節で述べたような WSN ソフトウェアを開発する最も一般的な手法は，特定のプラットフォームを選択し，そのプラットフォーム上で動作するソフトウェアを，DSL を用いて実装する方法である [62]．WSN 向けのソフトウェアプラットフォームは，プログラムを記述するための DSL と，その実行環境に当たる OS やミドルウェアの組合せとして，過去 10 年で数多く提案されてきた．図 2.2 は，WSN システム全体のアーキテクチャ

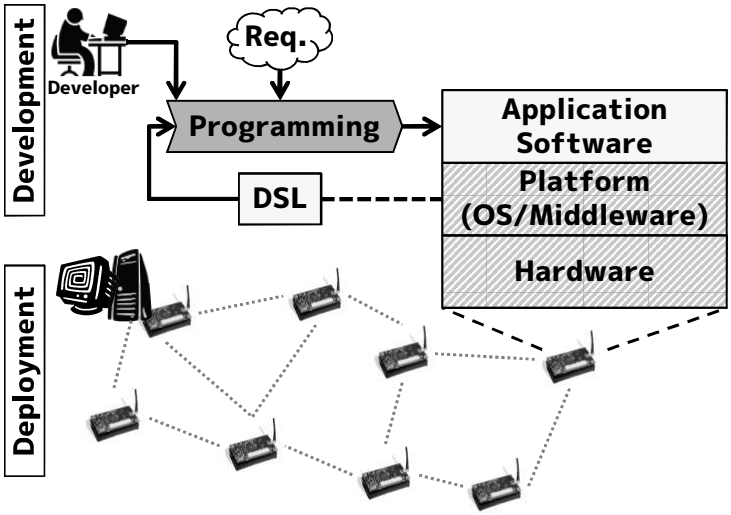


図 2.2: WSN システムのアーキテクチャと DSL を用いたソフトウェア開発の位置付け

表 2.3: WSN 向けプラットフォームの抽象度による分類

分類	属するプラットフォーム例
Dataflow-level	TinyDB [55], MacroLab [43], TikiriDB [49]
Group-level	Abstract regions [101], Regiment [65], RuleCaster [12]
Node-level	TinyOS [42] + NesC [34],Contiki[28], TeenyLIME [22]

と，アプリケーションソフトウェア部分に対する開発の流れを示している．WSN ソフトウェアはハードウェアや OS，非機能要求を考慮し決定され，固定的に運用されるプラットフォームに対して，プラットフォームに紐づく DSL を用いて開発される．すなわち，本研究における WSN ソフトウェア開発は，プラットフォーム上で動作するアプリケーションソフトウェアを開発する事を指す．

既存のプラットフォームは，サーベイ論文にて，プラットフォームが開発者に与える抽象化の視点から 3 段階に分類可能であることが知られている [62, 89]．本論文では，この 3 つの抽象度を Dataflow-level，Group-level，Node-level と呼ぶ．表 2.3 に，3 つの分類と各分類に属するプラットフォームの例を挙げる．

Dataflow-level に属するプラットフォームは，データ処理のみを記述する抽象化を提供する．TinyDB では，WSN 全体をデータベースとみなし，SQL を基とした形式で必要なデータとそのデータへの集約処理を記述する．すなわち，WSN 内における通信の詳細を捨象し，データ処理のみを記述することで通信設計，実装の難しさを低減している．MacroLab では，開発者は WSN 全体の計測データをベクトルに格納するよう指定し，データへの集

約処理や融合処理をベクトルの操作として記述する．TikiriDB は，TinyDB と同様，WSN をデータベースとみなす抽象化と SQL を元にした DSL を提供している．

Group-level に属するプラットフォームは，ノードグループの構成とマクロな動作を記述する抽象化を提供する．Abstract regions では，地理的に近く，同種のタスクを実行するノード集合を *region* と定義し，*region* の構成基準 (e.g. 距離) と *region* の動作を記述する．すなわち，個々のノードの詳細を捨象し，ノード集合に対するマクロな動作を記述することで分散処理設計，実装の難しさを低減している．また，Regiment は地理的に近いノード集合をグループとして扱い，このグループにおけるデータの計測とデータに対する処理を記述可能とするプラットフォームである．RuleCaster は，特定の状態にあるノード集合をグループとし，グループの次の状態への遷移を動作ルールとして記述可能としている．

Node-level に属するプラットフォームは．個々のノードの動作を記述する抽象化を提供する．NesC は，WSN 用 OS である TinyOS 上で動作するプログラムを記述する言語であり，個々のノードの動作を実装する．OS が提供する API を用いることでハードウェアに非依存なプログラムを記述可能である．Contiki は TinyOS, NesC と同様，個々のノードに対する処理をハードウェア非依存で記述させる抽象化を提供している．TeenyLIME は，NesC と同様個々のノードの動作を記述するが，ノード間で共有するメモリ空間を導入することで通信に関する記述の簡略化を実現する．

これら 3 つの抽象度間には，プラットフォーム上で可能な記述の簡潔さと，記述の柔軟さとの間にトレードオフが存在する．抽象度の高い Dataflow-level プラットフォーム上では，開発者は通信，タスク割当の設計を捨象した簡潔な記述により開発コストを低減可能である．しかしながら，捨象された設計はミドルウェアの固定的な設計の実行されるため，設計変更の柔軟さに難を持つ．一方抽象度の低い Node-level プラットフォーム上では，開発者は通信，タスク割当の設計を含む細やかな設計変更により詳細な品質改善が可能である．しかし，主たる機能性であるデータ処理と同時に通信やタスク割当といった分散処理を扱うことは，関心事の混在につながり設計，実装が複雑化する．次節ではこのトレードオフを 2.1 節で述べた HBM を対象とした開発を具体例にとり詳述する．

2.3.2 プラットフォームを用いた開発例

WSN 向けプラットフォームとして最も頻用されるのは個々のノードの動作を記述可能なノードプログラミングが可能なプラットフォームである [62]．このようなプラットフォームの代表としては NesC と TinyOS の組合せが挙げられる．

この NesC を用いて，2.1 節で述べた HBM を実現するソフトウェアを実装する場合，開

発者はデータ処理だけでなく，通信方式やタスク割当までノードの振舞として実現する．図 2.3，2.4 に，NesC を用いて記述した，データ計測や通信の記述が混在するコード例の一部を示す．これらのコードは，プログラムをどのノードに割り当てるか，すなわちタスク割当を考慮した上で実装される．ただし，建築物の異常検知処理はデータ受信後に別のコンピュータ上で行われているため図中では省略している．

図 2.3 は，データの計測を担うノードのプログラムである．このプログラムにはタイマーを用いた時間の管理，計測処理の呼び出し，パケットの作成やデータ送信に関する記述が含まれている．図 2.4 は，データの中継を担うノードのプログラムである．このプログラムでは，目的のデータを受信後そのまま送信を行なっているが，データの集約や融合を行う場合には計測データを送信してくるノードを把握し，計測ノードと集約／融合処理ノードとの協調動作としてプログラムを記述する必要がある．

また，これらのプログラムを記述する際には，開発者はルーティングプロトコルやデータ圧縮，暗号化などノード間の通信方式についても設計，記述する必要がある．更に，これらのプログラムは各ノードに対しどのようなタスクを割り当てるかも決定しつつ実装される．HBM の場合，品質改善のために，開発者は送受信時に圧縮を行う通信方式の設計や，ノードの配置に基づくタスク割当の設計を分散プログラム上で実現可能である．

このような通信の設定やノード毎の動作に関する記述により，開発者は品質に対する要求を満たすための詳細な品質改善が可能となる．しかしながら，ノードプログラミングではデータ処理や通信，タスク割当とった関心事が混在してしまう．また，通信方式の決定と実装には無線通信に関する知識を要し，個々のノードに対する実装は分散プログラムに関する経験やノウハウを要する．

このような開発の難しさを低減するため，抽象化を導入した数多くのプラットフォームが提案されている．前節にて挙げた Abstract regions や TinyDB がこれに当たり，WSN 内通信や個々のノードの詳細を捨象することで設計，実装の難しさを低減している．

図 2.5 に，TinyDB を用いて記述した，データ処理のみが含まれるコード例を示す．なお，異常検知のための処理はデータ収集後に行うため，NesC のコード例と同様，図 2.5 にこの記述は含まない．TinyDB では，`SELECT` 節にて要求するデータの種類を，`SAMPLING PERIOD` 節にて計測 (と送信) 間隔を記述する．また，各データへの集約処理がある場合には `SELECT` 節にてその処理を記述する (例：温度の平均 ＝ `AVG(temp)`)．このように TinyDB では，開発者はデータ処理のみを設計，記述する．しかしながら，TinyDB では通信時には圧縮は行わないこと，全ノードに同じタスクを割り当てることが DSL の実行環境に当たるミドルウェアにより固定されているため，これらの設計変更はできない．

このように，Dataflow-level のような抽象化を用いたプラットフォームでは，Node-level プラットフォームに比べ非常に簡潔な記述で WSN からセンサデータを取得することが可

```
...
// Time management
event void Timer.fired(){
    // Data sensing
    call Temperature.read();
}
...
event void Temperture.readDone(error_t result, uint16_t data){
    msg->temperature = data;
    // Data transmission
    call Collection.send(AM_BROADCAST_ADDR,
            &msg, sizeof(temperature_msg_t));
}
...
```

図 2.3: NesC での HBM 実装例 (計測・送信タスク)

能である．一方で，品質を改善すること，HBM においては計測データによって圧縮を実行することやノードごとに異なる役割を割り当てることは困難である．

2.3.3 プラットフォームを用いた開発と品質改善のための要件の対応

前述のようにトレードオフを持つプラットフォームを用いて開発を行う場合，開発対象ごとの品質改善に対する要件を全て満たすことはできない．表 2.4 は，各プラットフォームを用いた場合，品質改善のための各要件を満たすことができるかについてまとめている．

表 2.4 に示すように，いずれのプラットフォームを用いた場合でも，全ての要件を同時に満たすことはできない．特に，Dataflow-level，Group-level のプラットフォームを用いた場合，細やかな設計変更はプラットフォーム側が対応していないため，プラットフォームに変更を加えずに達成することは不可能である．従ってこれら 2 つのプラットフォームを用いて品質改善のための要件を満たすことは現実的ではない．そこで本研究では，Node-level プラットフォームを対象として開発を行う際に品質改善のための全ての要求を満たすことを目的とする．Node-level プラットフォームは細やか設計変更が可能なため，品質を高めることが重要視される WSN ソフトウェア開発では，現実的なソフトウェアに対する運用実績も比較的豊富である．

```
...
// Data receive
event message_t *Receive.receive(message_t *msg,
        void *payload, uint8_t len){
    if(len == sizeof(temperature_msg_t)){
        // Data transmission
        call Collection.send(...);
    }
}
...
```

図 2.4: NesC での HBM 実装例 (送受信タスク)

```
SELECT accel_x, accl_y, temp, humidity
FROM sensors
SAMPLING PERIOD 5000
```

図 2.5: TinyDB での HBM 実装例

2.4　WSN ソフトウェア開発のためのモデル駆動開発

既存研究として，ソフトウェア工学の分野における知見である MDD を WSN ソフトウェア開発に適用する研究が提案されている．MDD は，最終的な実装の抽象表現であるモデルを開発の中心とし，変換による詳細化を通して実装を獲得する開発手法である [41, 81]．MDD にて用いるモデルは DSML により記述され，DSML の文法はメタモデルによって定義される．MDD では抽象表現であるモデルと自動化された変換を用いることでソフトウェア開発の生産性の向上を図っている [33]．図 2.6 は，WSN システムのアーキテクチャと，ソフトウェア部分に対する MDD の位置付けを示している．また，MDD の一種である MDA は，利用するモデルを PIM，PSM として，プラットフォームを基準に明示的に分離する手法をとっている [64]．特に MDA では，プラットフォームを基準とした関心事の分離により，プラットフォーム間での移植の性向上を図っている [88]．MDD を WSN ソフトウェア開発に適用することで，開発の複雑さの低減による生産性向上と，プラットフォーム非依存なモデルの導入による移植性向上が試みられてきた．

既存研究の多くは Node-level プラットフォームを最終的な実行対象としているものの，手法ごとに MDD 中で用いる DSML の抽象度が異なる．Malavolta らのサーベイ論文 [57] では 16 の手法が取り上げられており，比較項目として各手法の DSML で記述可能なモデ

表 2.4: WSN ソフトウェアの品質改善のための要件とプラットフォームを用いた開発手法との対応関係

	初期実装獲得工程		品質改善工程	
	要件 **1.** 低コストでのプロトタイプ開発	要件 **2.** 低コストでの移植	要件 **3.** 細やかな設計変更	要件 **4.** 端的な設計変更の記述
Dataflow-level プラットフォーム	データ処理のみを設計，実装するため 達成可	特定プラットフォーム向け開発のため 達成不可	通信，タスク割当の捨象のため 達成不可	データ処理のみを設計，記述するため 達成可
Group-level プラットフォーム	データ処理と通信を設計，実装するため ある程度達成可	特定プラットフォーム向け開発のため 達成不可	通信のみ設計変更可能なため 達成不可	ノード集合のマクロな動作を記述するため ある程度達成可
Node-level プラットフォーム	データ処理，通信，タスク割当を全て設計，実装するため 達成不可	特定プラットフォーム向け開発のため 達成不可	通信，タスク割当の設計変更可能なため 達成可	データ処理，通信，タスク割当を全て設計，実装するため 達成不可

ルの抽象度 (Computational Scope) を挙げている．Malavolta らによれば，調査した手法のうち，半数の手法が個々のノードに対する動作，すなわち Node-level プラットフォーム上で動作するソフトウェアを直接表現するための DSML を用いた開発手法を提案している．また内 4 つの手法が同種のタスクを担うノード集合の動作，すなわち Group-level プラットフォーム上のソフトウェアに相当する DSML を，内 1 つの手法が WSN 全体の動作，すなわち Dataflow-level プラットフォーム上のソフトウェアに相当する DSML をそれぞれ提案していた．この様に既存の WSN 向け MDD 手法は，最終的な実行環境として Node-level プラットフォームを対象としている場合でも，提供する DSML の抽象度により 3 段階に分類可能である．

これらの手法では，DSML により記述されたモデルは変換，コード生成を通して Node-level プラットフォームで実行可能な実装へと変換される．生産性向上を目指す手法としては，Thang らの研究 [91] や Losilla らの研究 [54] がある．Thang らの手法では，個々のノードに割当てられるセンサや通信インターフェース，タスクを表現する Node-level DSML

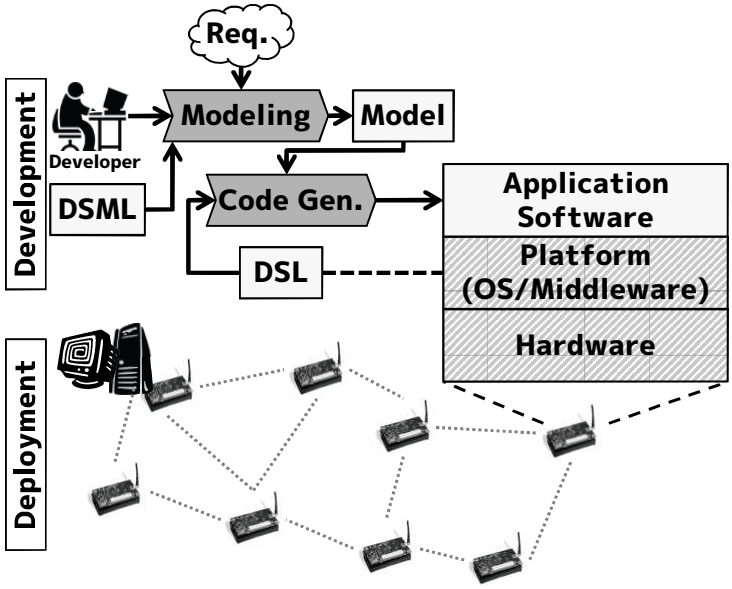


図 2.6: WSN システムのアーキテクチャと MDD の位置付け

を提案している．作成されたモデルはコード生成器を通して NesC のコードへと自動変換される．これにより，モデル上では通信，タスク割当の設計を表現しつつ，実装にかかるコストの低減を実現している．Losilla らの研究では特定の Node-level プラットフォームに依存しない WSN 用の DSML を定義している．この DSML は同種のタスクを担うノード集合の動作を表現しており，Group-level に対応する DSML であるといえる．記述されたモデルは NesC のコードへと自動変換することで実装コストを低減しており，ノード集合の動作というマクロな表現を用いることでモデルの表現を簡潔にしている．

また，対象とするプラットフォームごとにモデル変換規則，コード生成規則を切り替えることで，複数のプラットフォームへの移植開発を支援する研究も存在する．移植性向上を目指す手法としては，前述の Losilla らの研究や Mozumdar らの研究 [63]，Rodorigues らの研究 [74, 75] が挙げられる．これらの手法では提案する DSML で記述されたモデルを PIM とし，対象とするプラットフォームに応じた PSM，実装へと変換することで移植を支援する．Mozumdar らは，振舞モデルとしての PIM を用いる MDA フレームワークを提案しており，記述された PIM はシミュレーションとコード生成に利用可能としている．この PIM は個々のノードに対する記述可能，すなわち Node-level に対応する PIM であり，TinyOS と MANTIS [11] の 2 つの OS に対しコード生成が可能となっている．Rodrigues らは，ノード集合に対する動作を表現する PIM，すなわち Group-level に該当する PIM を定義し，これを用いる MDA フレームワークを提案している．Rodrigues らのフレーム

ワークでは，PIM は対象ドメインの専門家が，PSM はネットワークの専門家が用いることを想定している．また，PIM は TinyOS と SunSPOT (Squawk VM) [86] という 2 つの Node-level プラットフォームに対応する PSM，実装コードへと変換され，これによりプラットフォーム間の移植を実現している．

上述の様に，WSN 向けの MDD 手法は，モデルを用いることで生産性向上及び移植性向上に寄与している．しかしながら，2.2 節で述べた，開発対象ごとの品質改善を考えた場合に必要となる要件を全て同時に満たすことはできない．次節ではこの問題について詳述し，本論文が取り組む問題を示す．

2.5 WSN ソフトウェアの品質改善支援における問題

WSN ソフトウェア開発は Node-level プラットフォームを対象とする場合が多く，また本研究も Node-level プラットフォームへの開発を扱うが，開発対象ごとの品質改善を考える場合，その支援は十分とはいえない．表 2.4 にまとめたように，Node-level プラットフォームを対象とする場合，通信，タスク割当の設計変更が可能であるため，図 2.1 における品質改善工程に対する要件のうちの一つである “細やかな設計変更” は達成可能である．しかしながら，品質改善工程におけるもう一つの要件である “端的な設計変更” は，関心事の混在のため困難である．また，初期実装を獲得する工程における要件である，“低コストでの初期実装獲得” は，現状の開発手法では新規開発，移植開発の双方にて満たすことができない．

既存の WSN ソフトウェア支援に関する研究として，2.4 節で述べた MDD を用いる手法が挙げられるが，これらの手法を用いた場合でも品質改善のための 4 つの要件を同時に達成することは困難である．表 2.5 に MDD を用いた既存手法と，各種法が品質改善のための要件を達成可能であるかについてまとめた結果を示す．MDD 手法は，各手法が提供する DSML の抽象度により Dataflow-level MDD，Group-level MDD，Node-level MDD と表記する．

Dataflow-level MDD を用いた場合，DSML により記述されたモデルの簡潔さから “要件 1. 低コストでのプロトタイプ開発” や “要件 4. 端的な設計変更の記述” は達成するものの，その簡潔さのため “要件 3. 細やかな設計変更” を達成できない．Node-level MDD を用いた場合，通信，タスク割当を扱う DSML により “要件 3. 細やかな設計変更” は達成するが，関心事の混在によるモデルの複雑さから “要件 1. 低コストでのプロトタイプ開発” や “要件 4. 端的な設計変更の記述” は達成できない．Group-level MDD はこれら 2 つの中間に当たるが，通信設計のみを扱う DSML であるため “要件 3. 細やかな設計変更”

表 2.5: WSN ソフトウェアの品質改善のための要件と既存の MDD 手法との対応関係

	初期実装獲得工程		品質改善工程	
	要件 1．低コストでのプロトタイプ開発	**要件 2．低コストでの移植**	**要件 3．細やかな設計変更**	**要件 4．端的な設計変更の記述**
Dataflow-level MDD	データ処理のみを表現するため 達成可	データ処理のみを移植するため 達成可	通信，タスク割当の捨象のため 達成不可	データ処理のみを表現するため 達成可
Group-level MDD	データ処理と通信を表現するため ある程度達成可	抽象度が異なる PSM との乖離が大きいため 達成不可	通信のみ表現可能なため 達成不可	マクロな動作を表現するため ある程度達成可
Node-level MDD	データ処理，通信，タスク割当を全て表現するため 達成不可	抽象度が異なる PSM との乖離が大きいため 達成不可	通信，タスク割当を表現可能なため 達成可	データ処理，通信，タスク割当を全て表現するため 達成不可

は達成できず，またデータ処理と通信を同時に扱うため “要件 1. 低コストでのプロトタイプ開発” や “要件 4. 端的な設計変更の記述” も達成できない．これは，2.3.2 節，2.3.3 節で述べたトレードオフとほぼ同様である．

更に，Group-level MDD，Node-level MDD を用いる場合，異なる抽象度のプラットフォーム間での移植では “要件 2. 低コストでの移植” を達成できない．同抽象度のプラットフォーム同士での移植を対象とする場合は，PIM と PSM が持つ設計情報が同程度であるため移植を実現できる．しかしながら，異なる抽象度のプラットフォームを対象とした移植では，PIM と PSM が持つ設計情報に齟齬が生じうる．例えば，Dataflow-level プラットフォームを対象とした既存ソフトウェアの移植を考える場合，PSM を含む既存成果物はデータ処理設計に関する情報のみを持つ．この際，Group-level MDD や Node-level MDD で用いる DSML 上，すなわち PIM では，データ処理のみでなく通信設計やタスク割当設計を含むため，PSM と PIM 間に乖離が生じる．移植を達成し実行可能な実装を獲得するためには，不足する設計情報は開発者により決定，補完され，PIM に反映されなければならない．このような人手での意思決定，補完作業の結果，Group-level MDD，Node-level MDD では “要件 2. 低コストでの移植” を達成できない．

このように既存手法では全ての要件を同時に達成することは困難である．これに対し本

研究では，4 つの要件を全て満たす，開発対象ごとの品質改善支援を実現するソフトウェア開発手法の構築を目指す．既存の MDD 手法は，いずれも単一の抽象度のモデルのみを用いており，異なる抽象度間に存在するトレードオフのために 4 つの要件を達成できない．そこで本研究では，3 段階の抽象度のモデルを併用，使い分けることで既存研究における問題の解決を図る．

図 2.7 に，提案手法の全体像を示す．提案手法では 3 段階の抽象度の DSML と，DSML で記述されたモデル間を接続する変換規則からなる．初期実装獲得工程においては，新規開発に対してはデータ処理設計のみを対象とすることで低コストでのプロトタイプ開発を試みる．また，移植開発における初期実装獲得工程では，3 段階のモデルを PIM とし，移植元プラットフォームと同抽象度の PIM を通して移植を行う．不足する設計情報は変換規則により自動で補完することで人手によるコストを低減し，多様なプラットフォームからの低コストでの移植を試みる．品質改善工程においては，3 段階の抽象度から任意の抽象度を選択し設計変更を施すことで，細やかな設計変更と端的な変更の記述の両立を試みる．

図 2.7 に示す全体像を実現する基盤となる多段階 MDD フレームワークを第 3 章にて提案する．このフレームワークを新規開発に適用した開発手法を第 4 章にて，移植開発に適用した開発手法を第 5 章にてそれぞれ提案する．これにより，開発対象ごとの品質改善に求められる 4 つの要件を全て満たす開発手法の構築を行う．

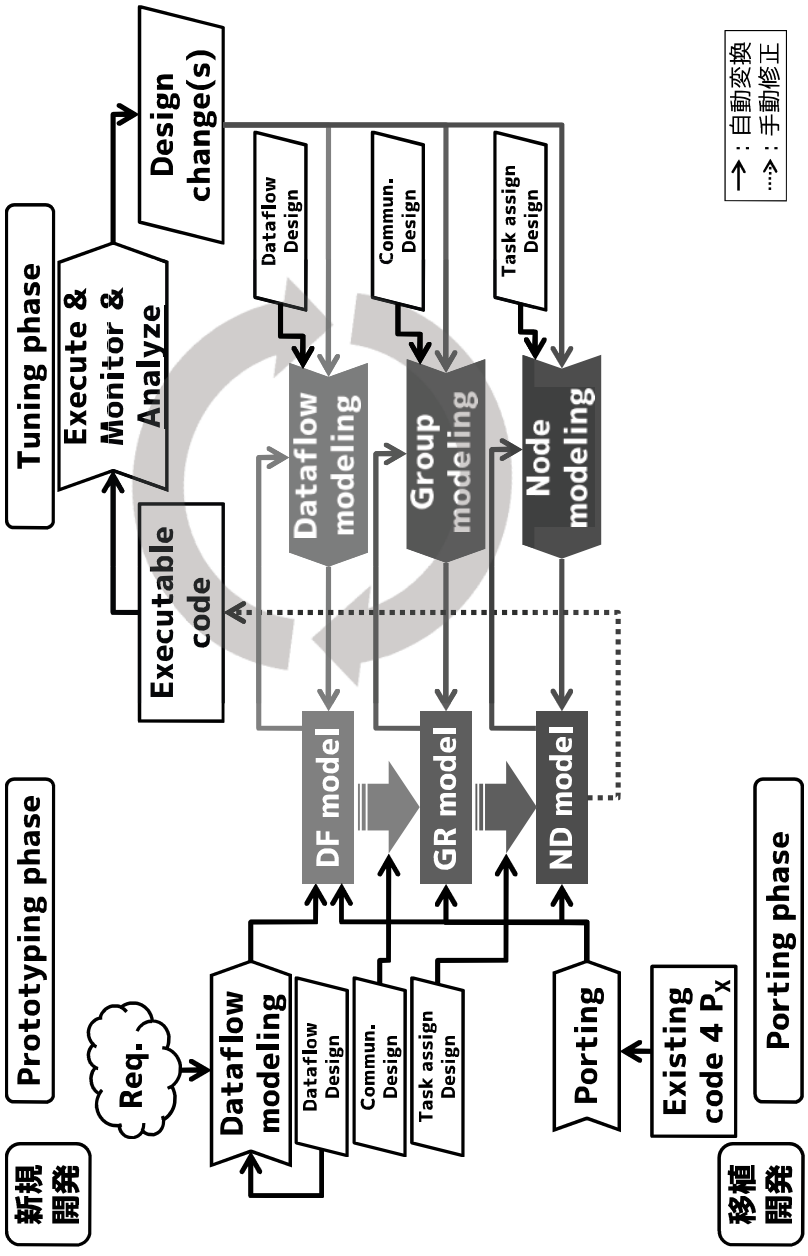


図 2.7: 提案手法による開発プロセスの全体像

第3章　無線センサネットワークソフトウェアのための多段階モデル駆動開発フレームワーク

WSN ソフトウェア開発における品質改善のための 4 つの要件を満たすため，本研究では複数の DSML を併用する多段階 MDD フレームワークを提案する．本フレームワークは，図 3.1 に示すように，3 種の DSML，3 種のモデル間をつなぐ変換規則，コード生成器から構成される．

品質改善のための初期実装獲得工程においては，新規開発では “要件 1. 新規開発における低コストでのプロトタイプ開発が可能であること” を，移植開発では “要件 2. 移植開発における多様な WSN プラットフォームからの低コストでの移植が可能であること” を満たす必要があった．また品質改善工程においては，“要件 3. 通信設計，タスク割当設計を含む細やかな設計変更が可能であること” と “要件 4. 品質改善のための設計変更を端的に記述可能であること” を満たす必要がある．

これに対し本フレームワークを用いることで，新規開発の初期実装獲得工程においては，データ処理設計のみを扱い，変換により下位のモデル，実装を生成することで低コストでのプロトタイプ開発を達成する．また移植開発の初期実装獲得工程においては，移植元 PSM を，移植元プラットフォームと同抽象度の PIM へと変換し，フレームワーク内の変換により不足する設計情報を自動で補完することで低コストでの移植を実現する．品質改善工程においては，3 段階の抽象度のモデルから任意の抽象度を選択的に用いることで細やかな設計変更と端的な変更の記述を両立する．

新規開発に対しては，本フレームワークを用い，データ処理設計，通信設計，タスク割当設計の分離を図り，工程ごとに 3 つの DSML を使い分ける開発プロセスを提案する．多段階抽象度の併用による開発プロセスにより，低コストでのプロトタイプ開発と細やかで端的な設計変更による品質改善を両立する開発の支援を行う．この開発プロセスについては第 4 章にて詳述する．

移植開発に対しては，3 段階の抽象度の PIM を併用し，移植元プラットフォームに対応

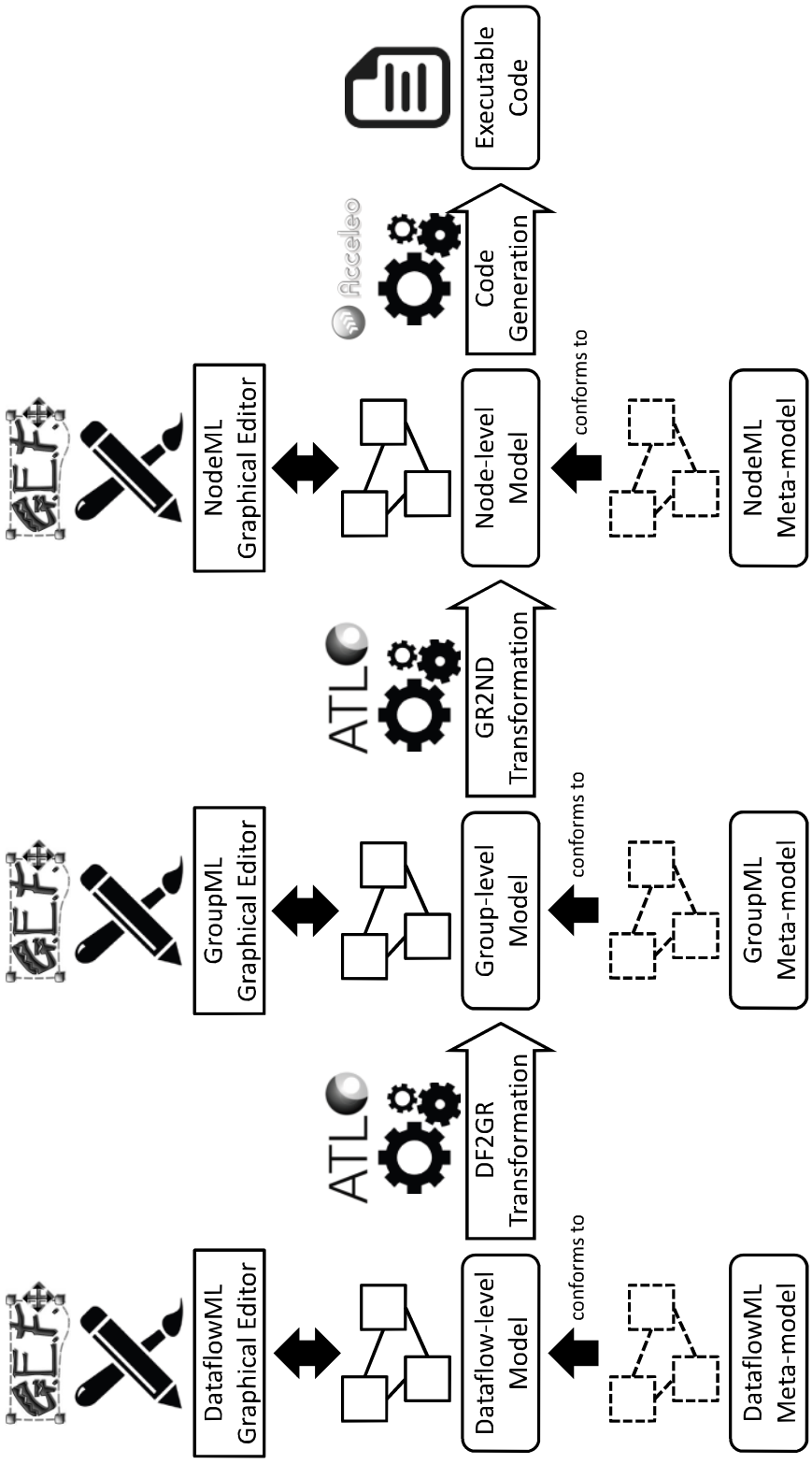


図 3.1: 提案フレームワークの全体像と実装概要

する抽象度のPIMを用いる開発手法を提案する．多段階抽象度のPIMの併用と自動変換により，多様なプラットフォームからの低コストでの移植と，移植後の品質改善を実現する開発支援を行う．この開発手法については第5章にて詳述する．

以降の各節では本フレームワークの構成要素について詳述する．3.1節では3つのDSMLについて，3.2節ではDSMLにて記述されたモデルをPIMとみなし，PIM間をつなぐ変換規則について記す．3.3節では，DSMLと変換規則を含む本フレームワークのツール実装の詳細について述べる．また，コード生成とプラットフォーム間移植に必要となる，PIM・PSM間の変換とコード生成器の実装についても述べる．

3.1 DSMLの設計

本フレームワークでは，トップダウンには分散ソフトウェア開発におけるデータフロー設計，タスクアーキテクチャ設計，詳細設計の3つの工程を，ボトムアップには既存のWSN向けプラットフォームの分類 [62] に基づき3つのDSMLを定義する．DSMLは，WSNソフトウェアのデータフローを表現するDataflow-level，ノード集合の構成と処理を表現するGroup-level，個々のノードの役割とタスクを表現するNode-levelの3つを定義した．これらのDSMLは，WSNソフトウェアとして最も典型的な，定期的なデータ計測・送信を行うWSNソフトウェアを開発可能なよう設計した．またDSML上での記述は特定のプラットフォームに非依存となるよう設計している．すなわち，本DSMLで記述されたモデルはPIMとみなすことができる．

本節では以下で個々のDSMLについて詳述するが，まず各DSMLの記述例の対象として用いるWSNソフトウェアについて述べる．DSMLの記述例として，室内温度の取得を実現するWSNソフトウェアを開発する場合を考える．対象ソフトウェアは，特定の部屋に配備されたセンサノードから温度データを取得し，それらの平均値を定期的に取得するものとする．以下の各節では，このWSNソフトウェアを実現するモデルを記述例にとる．

3.1.1 DataflowML

DataflowMLは，WSNソフトウェアの主たる機能要求である，ネットワークに非依存なデータフロー設計を記述するためのDSMLである．データフローを表現するためには，データの出力を行うデータソース，得られたデータに対して集約または融合処理を行う中間処理地点，最終的にデータを集めるデータシンクの3つと，これらの要素を接続し，データの移動経路を表すリンクが必要となる．本研究では，集約処理とは複数の同種の

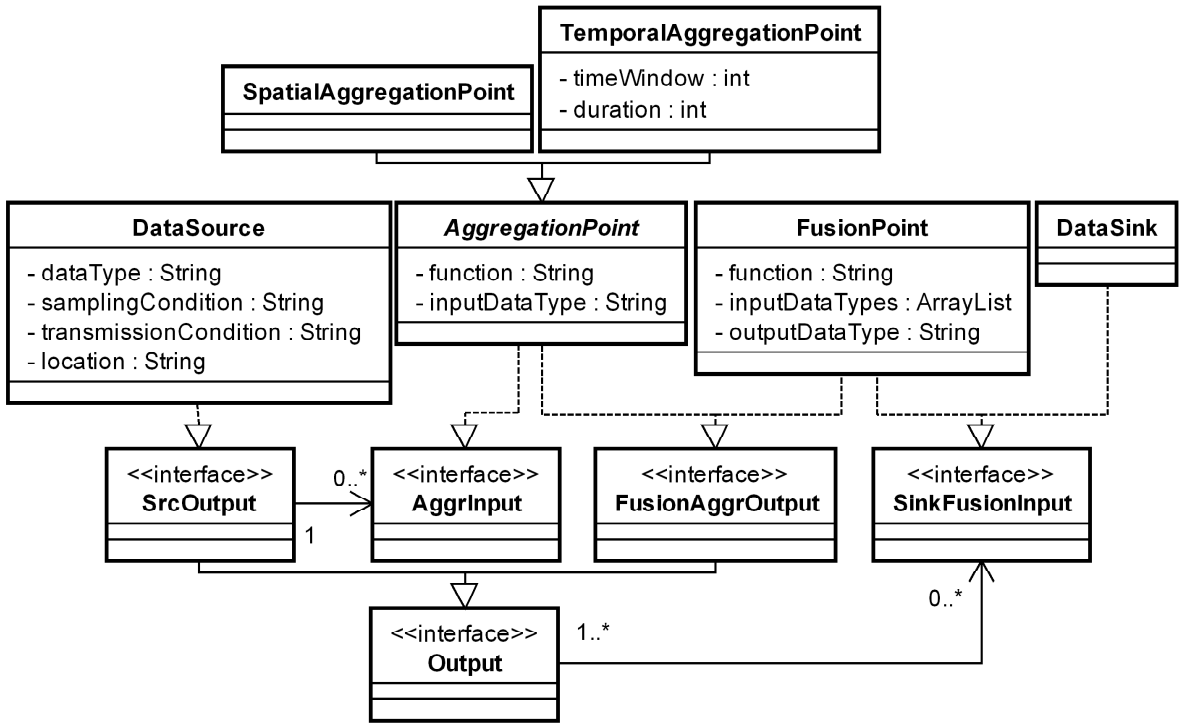


図 3.2: DataflowML のメタモデル

データを入力とし，入力から一つの代表値を出力する処理を，融合処理とは複数の異種のデータを入力とし，一つの代表値を出力する処理をそれぞれ指す．

このようなデータフローを記述可能とする DataflowML の定義に当たるメタモデルを図 3.2 に示す．DataflowML では，データの出力を行う `DataSource`，得られたデータに対して集約または融合処理を行う `AggregationPoint` と `FusionPoint`，最終的にデータを集める `DataSink` と，これらの要素を接続しデータの流れを表現するリンクを構成要素としている．集約処理については，WSN が時空間的に分散するデータを扱うことから，時系列の計測値から代表値を得る処理 (`TemporalAggregationPoint`) と，空間的に分散する計測値から代表値を得る処理 (`SpatialAggregationPoint`) の二種類が必要と考え，各々に対応する要素を用意した．また，メタモデルでは各要素に必要なパラメタも定義している．例えば，データソースにおいて計測に必要な各種情報 (計測データの種類，計測間隔，送信間隔，計測場所) は，`DataSource` の属性 (`dataType`, `samplingCondition`, `transmissionCondition`, `location`) に対応している．DataflowML を用いることで，WSN ソフトウェアで実現したい機能性を，データフローとして簡潔に記述可能である．

平均温度監視を実現する WSN ソフトウェアに対し，DataflowML を用いて記述したモ

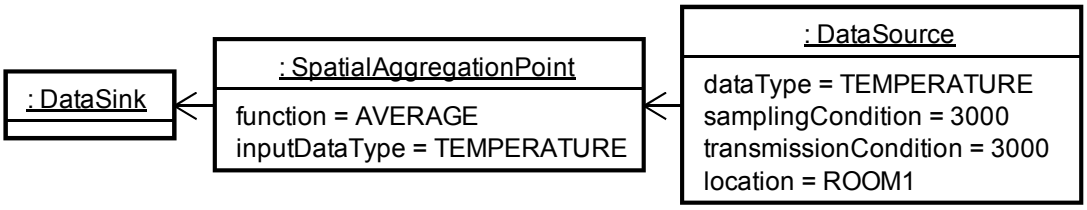


図 3.3: 平均温度監視における Dataflow-level モデルの記述例

デル (以降，Dataflow-level モデル) の例を図 3.3 に示す．このモデルでは，DataSource により 30 秒間隔で温度データを計測，送信すること，SpatialAggregationPoint により同時刻に計測された温度データの平均値を計算することが，一つのデータフローとして示されている．このように，Dataflow-level モデルでは，WSN ソフトウェアが実現するデータフローを非常に簡潔に表現可能である．

3.1.2 GroupML

GroupML は，タスクアーキテクチャ設計を，階層的なノード集合の構成とグループ間の通信として記述するための DSML である WSN ソフトウェアにおけるタスクアーキテクチャは，WSN が達成すべきデータフローをネットワーク上で実現するためのタスクに分割し，タスク間での通信方式を設計した結果である．これらを表現するため，同じタスクを担うノード集合をグループとし，各グループはリーダとメンバから構成されるものとした．本研究が対象とする WSN ソフトウェアのネットワーク上でのデータの流れは，計測ノードからマルチホップ通信を通し，必要に応じて集約・融合処理を経て段階的に収集地点となるノードに届けられる，というものである．リーダ・メンバ型のグループでは，データの流れがメンバからリーダとなることが主であり，リーダはメンバから受け取ったデータをより上位のグループのリーダへと流すため，対象とする WSN ソフトウェアのデータの流れと親和性が高いと考えた．よって，本研究ではリーダ・メンバ型のグループを用いたモデル化を対象とする．

このようなタスクアーキテクチャを記述可能とする GroupML のメタモデルを図 3.4 に示す．GroupML では Group を最も基本的な要素とし，各グループは，リーダとして WSN 全体のデータを集める Sink 或いはグループ内のデータを集める LeaderNode を，メンバとして子要素の Group 或いは計測処理を担う MemberNodes をそれぞれ持つ．リーダは集約・融合処理を担うため，これを Operator として定義している．更に，リーダ・メンバ間の通信方式を Communication という要素として含む．

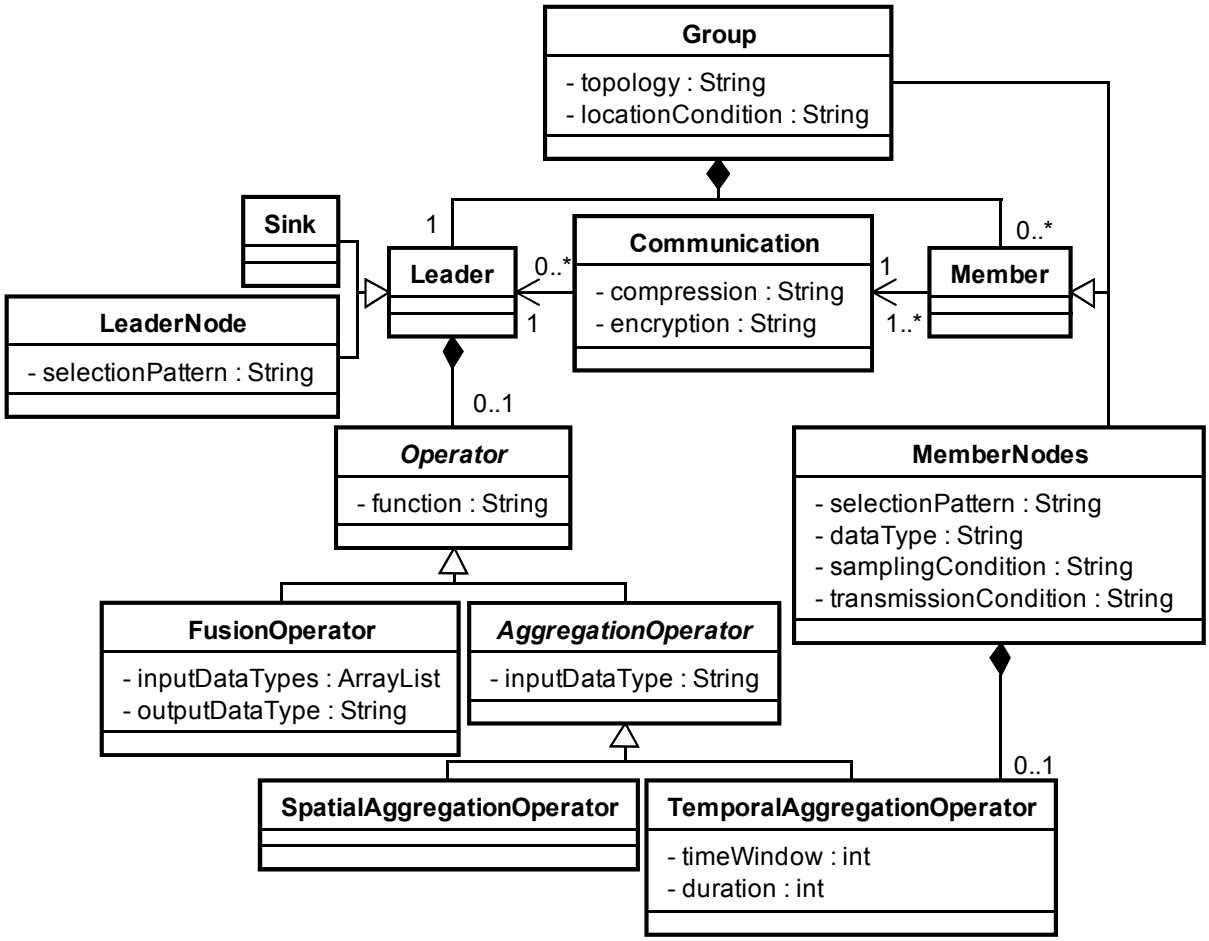


図 3.4: GroupML のメタモデル

また，品質改善のためには，グループ内のネットワークトポロジやリーダ・メンバ間での通信方式，リーダ・メンバの選択条件，グループの構成が変更可能である必要がある．表 3.1 に示す様に，GroupML ではモデルの属性やグラフ構造を修正することでこれらの設計を表現，変更可能とした．例えばデータ圧縮を用いない場合は `Communication` の属性 `compression` の値を `NONE` に，用いる場合は圧縮アルゴリズムに対応する語彙に設定することで表現できる．また，グループクラスタリングを用いる場合はモデルのグラフ構造を階層的にする，用いない場合は全ての要素を一つの `Group` 内に含めることで表現できる．GroupML を用いノードグループの構成を記述することで，グループ単位でのタスク分割と，グループ間／グループ内での通信方式を記述可能である．

図 3.5 に，平均温度監視を実現する WSN ソフトウェアを，GroupML を用いて記述し

表 3.1: Group-levelにおける設計空間

Group-levelでの設計	対応するモデル中の要素
ネットワークトポロジ	`Group.topology`
データ圧縮	`Communication.compression`
暗号化	`Communication.encryption`
リーダの選択条件	`LeaderNode.selectionPattern`
メンバの選択条件	`MemberNodes.selectionPattern`
データ処理地点	`Operator`の割り当て
クラスタリング	`Group`の階層構造

たモデル (以降，Group-levelモデル) の例を示す．このモデルは，WSNソフトウェアは3つの階層的なグループからなり，下位の`MemberNodes`・`Group`が上位へとデータを送る様子を表している．最下位の`MemberNodes`が30秒間隔で温度データを計測，送信し，中間グループの`LeaderNode`が`SpatialAggregationOperator`により温度データの平均値を計算するというデータフローが，階層的なグループの構成として表現されている．また，各グールプ内ではツリー型トポロジを用いること (`topology=TREE`)，リーダ・メンバ間の通信では圧縮・暗号化ともに用いないこと (`compression=NONE`, `encryption=NONE`) といった通信方式の設計が，対応する属性により表現されている．更に，リーダの選択条件は特に指定されておらず (`LeaderNode.selectionPattern=ANY`)，メンバとしての処理は全てのノードに割り当てられる (`MemberNodes.selectionPattern=ALL`) といった，タスク割当に関するマクロな条件も表現されている．

3.1.3 NodeML

NodeMLは，詳細設計を個々のノードの動作として記述するためのDSMLである．WSNソフトウェアは，最終的には個々のセンサノード上で実行される．各ノードの動作は，タスクアーキテクチャ設計にて分割されたタスクを個々のノードに割り当てることで決定される．また，幾つかのタスクをまとめることである種の役割を表すと考えられる．ノードに割り当てられるタスクや役割は，GroupMLで表現されるノード集合の動作を詳細化したものであるためであるため，データの収集地点となるベースステーションとしての役割，データの操作と中継を担うリーダとしての役割，データの生成を担うメンバとしての役割の3種類が考えられる．したがって，詳細設計の記述には，WSNを構成するノードと，ノードに対して割り当てられるタスク及び役割が必要となる．

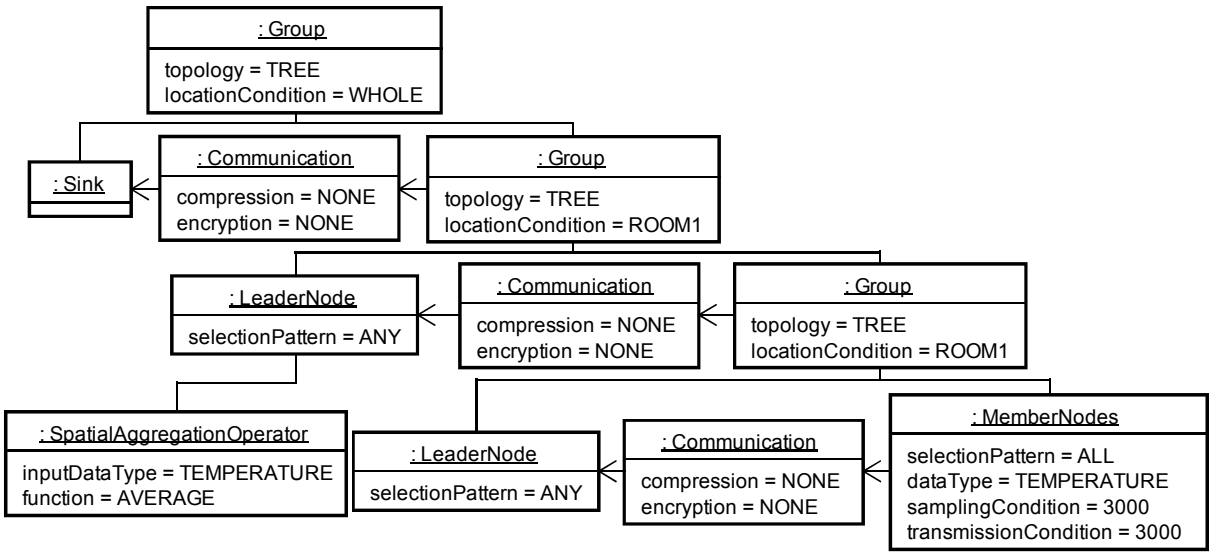


図 3.5: 平均温度監視における Group-level モデルの記述例

このようなタスク割当を記述可能とする NodeML のメタモデルを図 3.6 に示す．NodeML では，ノードへのタスク割当を表現するため，物理的なノードを表す概念 (Node) と，各ノードに割り当てるタスクを表す概念 (Task) を構成要素とした．Task の種類としては，データ計測の表す SamplingTask，データ処理を表す OperationalTask，データ送信を表す SendingTask，データ受信を表す ReceivingTask を定義した．また，各ノードは複数のタスクを持つことで一つの役割を持つため，これを Role として NodeML の構成要素とした．例えば，センサデータの計測タスクと送信タスクを合わせることでノードグループのメンバとしての役割 (MemberRole) を持つと考えられる．その他，リーダとしての役割を意味する LeaderRole，データを集める役割を意味する BaseStationRole を記述要素とした．NodeML を用い，タスクと役割を個々のノードに割り当てることで，個別ノードの動作をタスク割当として記述可能である．また，SendingTask の属性 protocol など，通信に関わるタスクの属性を決定することで，通信方式についても記述可能である．

図 3.5 に，平均温度監視を実現する WSN ソフトウェアを，NodeML を用いて記述したモデル (以降，Node-level モデル) の例を示す．このモデルは，ソフトウェアが 4 つの Role を，4 つの Node へ割り当てることで実現されることを表している．各 Role は複数の Task からなり，MemberRole が温度データの計測タスク (SamplingTask) とリーダへの送信タスク (SendingTask) を担うことが記述されている．また，2 つの LeaderRole が温度データの受信タスク (ReceivingTask) と送信タスク (SendingTask) によるデータの中継と，平均値への集約タスク (SpatialAggregationTask) を担い，BaseStationRole がすべての

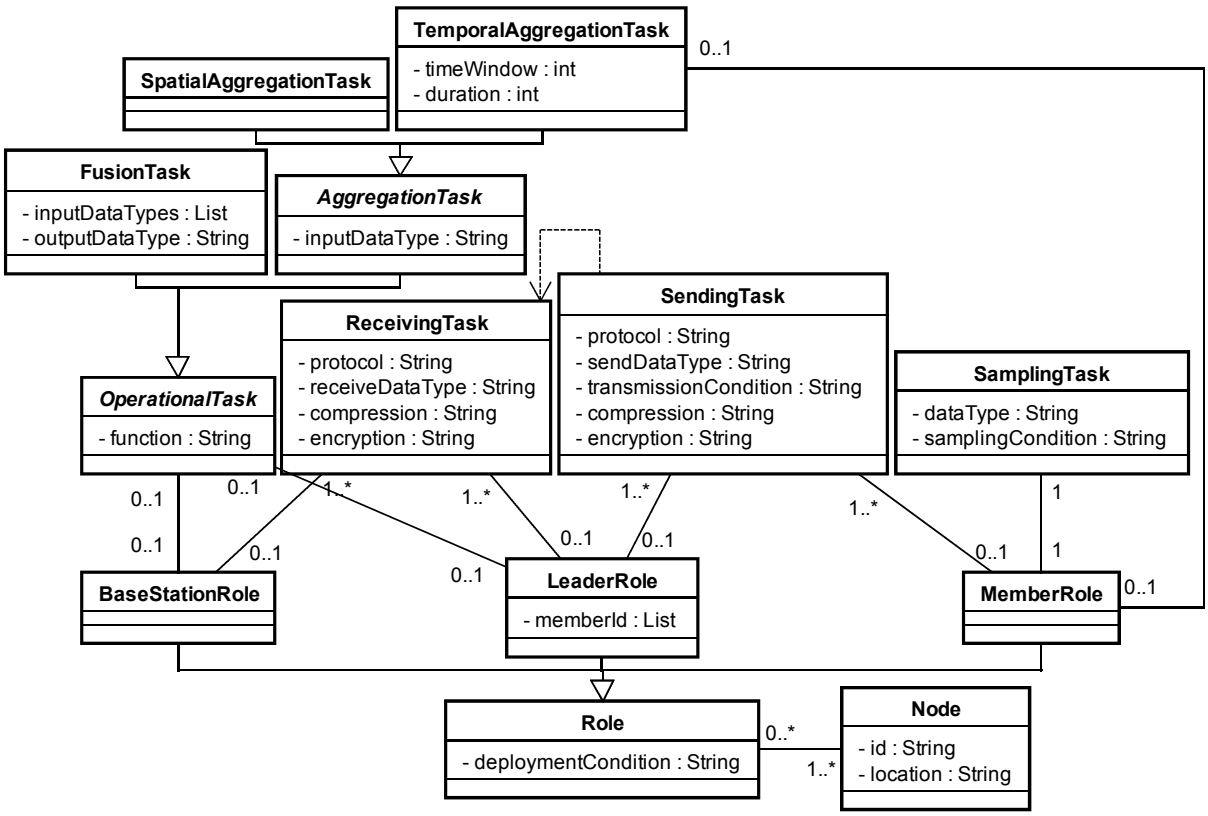


図 3.6: NodeML のメタモデル

温度データ受信を担うことをそれぞれ表している．各 Role 内の送受信タスクのつながりによりデータフローを実現している．また，ツリー型のルーティングプロトコルである CTP(Collection Tree Protocol) [36] を用いること (protocol=CTP) などの通信方式の設計は送受信タスクの属性，タスク割当の設計は Role と Node との間の関連により表現されている．更に，各 Role はそれぞれ 1 つのセンサノードに割り当てられていることが記述されている．

3.2 PIM 間の変換規則の設計

本 MDD フレームワークでは Dataflow-level から Group-level，Group-level から Node-level への 2 つの PIM 間の変換規則を定義した．この自動変換により，複数のモデルを扱う際に必要となる，詳細化における上位から下位への設計の伝播を実現する．異なる抽象

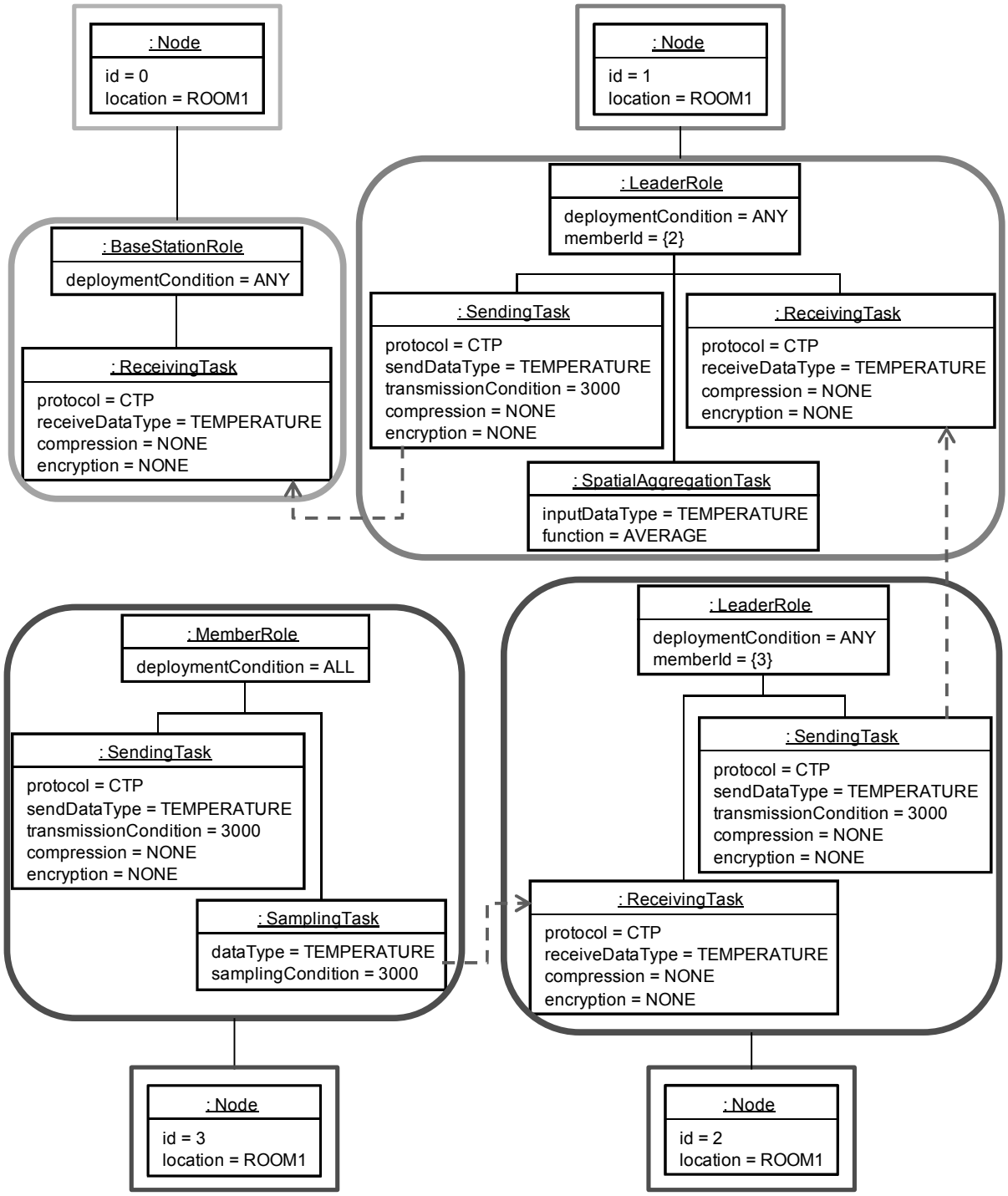


図 3.7: 平均温度監視における Node-level モデルの記述例

度をまたがる変換は，これまでは特定の DSL ごとに行われてきたが，本研究では各抽象度の DSL の抽象表現としてのモデルに対する変換を考えることで，抽象度間の関係を整理し，変換規則として定義した．特に，既存の DSL において，Dataflow-level，Group-level に属する DSL での実装は，直接 Node-level の DSL へと変換され実行される．このため，これまで Dataflow-level から Group-level への変換については考えられていないが，本研究では新たにこの変換に対する規則を定義，実現した．

また，詳細化の過程では，上位のモデルには含まれないが下位のモデルには含まれる設計を付与する必要がある．例えば Dataflow-level モデルを Group-level モデル，Node-level モデルへと変換する場合，通信方式，個々のノードへの役割の付与などを決定する必要がある．これらの設計を，WSN の実行前に人手で決定することは困難であるため，定義した変換規則ではこれらの設計に対し初期値を与えている．すなわち，定義したモデル変換規則は，各モデルに含まれる要素を対応付ける規則と，抽象度の低いモデルに新たに現れる情報に対して初期値を与える規則の 2 つからなる．

本節では 2 つの変換規則の定義と，例を用いた変換の詳細について述べる．具体例としては前節と同様，平均温度監視の WSN ソフトウェアを用いる．具体例となる変換の全体像を図 3.8 に示す．

3.2.1 Dataflow-level モデルから Group-level モデルへの変換規則

まず，Dataflow-level から Group-level へのモデル変換では，ネットワーク非依存なデータフローをネットワーク上でのノード集合の動作に変換する．この変換では，入力モデルの `DataSink` から処理を開始することを想定し，Algorithm 1 に示すようにリンクを辿って要素を探索，発見した要素に応じて Group-level の要素を生成していく．

この変換において，表 3.1 にて示した設計は Dataflow-level モデルには現れない要素である．例えば，グループ内トポロジやリーダ･メンバ間の通信などの要素は，Dataflow-level モデルには現れない．Dataflow-level における `AggregationPoint` や `FusionPoint` は，ネットワーク上での実行を考える場合，ネットワーク内で実行するか，データ収集後に実行するかという選択が存在する．

この様な Group-level モデル生成にあたり決定する必要のある設計は変換規則にて初期設定を与えている．通信については，グループ内では初期値としてツリー型のトポロジを用いて通信し，圧縮や暗号は行わないものとした．また，グループリーダ・メンバの選択については，リーダの指定はせず，WSN 全体をメンバとして用いるものとしている．集約・融合といったデータ処理の実行場所については，初期値としてネットワーク内で実行する設定を与える．これら与える初期設定の一覧を表 3.2 に示す．これら初期設定は

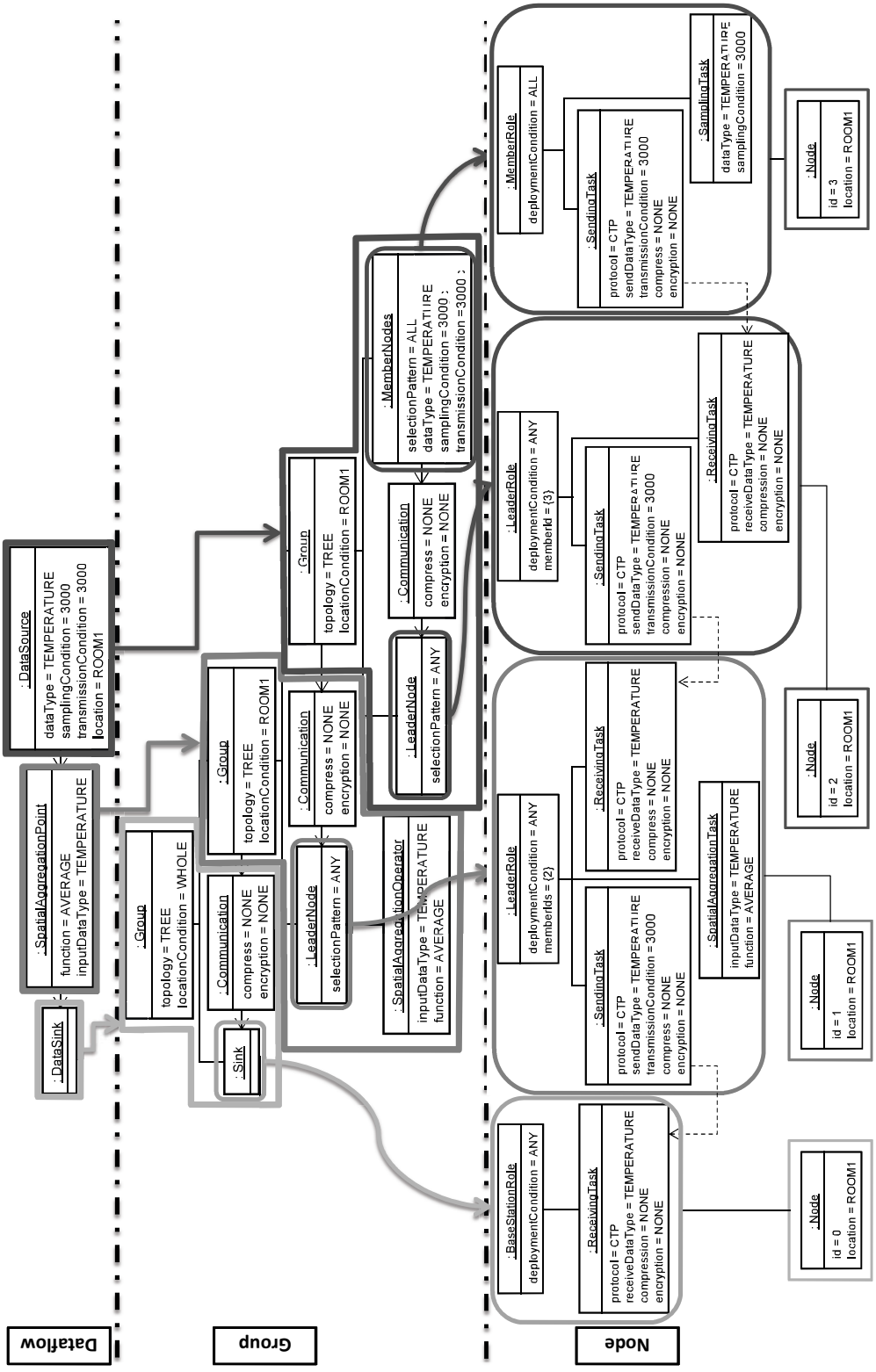


図 3.8: 平均温度監視における変換例の全体像

Algorithm 1 DataflowLevel2GroupLevel

```
1: function generateGroupModel(dataflowModel)
2:     groupModel ←makeGroup(dataflowModel)
3:     return groupModel
4: end function
5: function makeGroup(inputModel)
6:     head ← inputModel.root
7:     group ← vertex with Group
8:     if head.equals(DataSource) then
9:         group.append(LeaderNode)
10:        group.append(MemberNodes)
11:    else
12:        if head.equals(DataSink) then
13:            group.append(Sink)
14:        else if head.equals(FusionPoint) then
15:            group.append(LeaderNode with
                   FusionOperator)
16:        else if head.equals(AggregationPoint) then
17:            group.append(LeaderNode with
                   AggregationOperator)
18:        end if
19:        for each child in head.children do
20:            group.append(makeGroup(child))
21:        end for
22:    end if
23:    return group
24: end function
```

表 3.2: Group-level への変換時の初期設定

Group-level での設計	**初期設定**
ネットワークトポロジ	ツリー型
データ圧縮	なし
暗号化	なし
リーダの選択条件	Any
メンバの選択条件	All
データ処理地点	ネットワーク内
クラスタリング	あり

Group-level にて取りうる設計上の選択肢の一つであり，例えば異なるトポロジを用いる場合には，変換後に手動で自動生成されたモデル上で与えられた初期設定を変更する必要がある．

変換規則にて要素間の対応関係と未決定事項の初期設定を与えることにより，Dataflow-level から Group-level への自動変換を実現した．ただし，Algorithm 1 は入力モデル探索手順の概要を示しており，細かな属性の対応付けや初期設定の付与については割愛している．

この変換を図 3.8 における平均温度監視を例に詳述する．変換では入力となる Dataflow-level モデル (図 3.8 上部) を `DataSink` から探索する (Algorithm 1 行 1-2)．まずは現在探索中の入力モデル内のルート要素を取得し (行 6)，変換先の情報を保持するための `Group` を作成する (行 7)．この `Group` の作成の際には，手動変換の場合は人手でトポロジの属性 `topology` の値を決定する必要があるが，本自動変換ではこの属性に，表 3.2 に示す初期設定値を与える．その後の処理は探索中の要素に応じ分岐する．まずは `DataSink` を扱うため，これに対応する `Sink` を行 7 で作成した `Group` に追加する (行 12-13)．続いて入力モデルの下位へと探索を進めるため，下位の要素郡の取得とこれを用いた `Group` の作成を行う (行 19-21)．この例では `DataSink` に接続されている `SpatialAggregationPoint` 以下のモデル要素を取得し，再帰的に makeGroup を呼び出す (行 20)．

以後は前述の `DataSink` の場合と同様，モデル中のルート要素を取得 (行 6)，`Group` を作成 (行 7)，`SpatialAggregationPoint` に対応する `LeaderNode` と `SpatialAggregationOperator` を `Group` に追加する (行 16-18)．これは，自動変換規則では，データ処理をネットワーク内で実行するという初期設定を与えているためである．この再帰呼び出しにより入力モデルを下位へ探索していく．この手順を繰り返し，入力モデルを `DataSink` から下位へと探索し，対応する Group-level モデルの要素を生成していく．リーダ作成後は `DataSource`

を入力として makeGroup を呼び出す．

ルート要素が DataSource である場合には，LeaderNode と，DataSource に対応する属性を持った MemberNodes を Group に追加し (行 8-10)，このグループを返り値として makeGroup を終了する (行 23)．これは，Dataflow-level モデルにおいては DataSource より下位に接続される要素が存在しないためである．この際，LeaderNode，MemberNodes の割当条件である属性 selectionPattern や，通信に関する要素 Communication の属性には，表 3.2 に示す初期設定を与える．makeGroup の返り値は，呼び出し元の makeGroup 内で作成された Group に子グループとして追加される (行 20)．このように makeGroup の返り値を Group に追加していくことで，再帰的に階層的なグループを生成する．この再帰的な実行の結果，図 3.8 中部に示す Group-level モデルが自動変換により得られる (行 3)．このようにして，Dataflow-level モデルが持つ情報を対応付けつつ，足りない情報を自動で付与する変換規則により，Group-level モデルの自動生成を実現する．

3.2.2 Group-level モデルから Node-level モデルへの変換規則

次に，Group-level から Node-level への自動変換では，ノード集合の動作を個々のノードの動作に変換している．この変換においては，まず Group-level モデルの構成要素を，WSN ソフトウェアを実現するために必要となる役割とタスクに変換する．変換では，入力モデルの最上位の Group から処理を開始し，Algorithm 2 に示すように関連する要素を探索，発見した要素を対応する Node-level の Role を生成する．例えば，Group-level における MemberNodes は，Node-level における MemberRole と，この役割を実現する計測タスク (SamplingTask) と送信タスク (SendingTask) に変換される．生成された役割とタスクを，WSN を構成するノードに割り当てることで Node-level モデルを生成する．役割とタスクをノードに割り当てるためには，実際の WSN を構成するノードに関する情報が必要であるため，変換時にはネットワーク情報を与える．ネットワーク情報とは，対象 WSN に存在するノードの ID や場所，任意のプロファイルに関する情報を指す．例えば，抽出した MemberRole の配備条件が ALL だった場合，全ての Node にこの MemberRole を割り当てる．このネットワーク情報と，抽出した役割・タスクとを配備条件に基づいてノードに割り当てることで Node-level モデルへの自動変換を実現した．Algorithm 2 は Algorithm 1 と同様，入力モデル探索手順の概要を示している．

この変換について，図 3.8 の平均温度監視における変換を例に詳述する．変換では入力となる Group-level モデル (図 3.8 中部) にて，最上位の Group から makeRole を呼び出し，探索を開始する (Algorithm 2 行 1-2)．まず makeRole の入力モデルを調べ，MemberNodes かそうでないかにより処理を分岐する (行 6, 8)．その後，MemberNodes である場合は対応

Algorithm 2 GroupLevel2RoleSetAtNodeLevel

```
1: function generateRoleSet(groupModel)
2:     roleSet ←makeRole(groupModel, roleSet)
3:     return roleSet
4: end function
5: function makeRole(inputModel, roleSet)
6:     if inputModel.root.equals(MemberNodes) then
7:         roleSet.add(MemberRole with
               SamplingTask & SendingTask)
8:     else
9:         group ← inputModel.root
10:        if group.leader.equals(Sink) then
11:            roleSet.add(BaseStationRole with
                   ReceivingTask for each
                   member in group.members)
12:        else if group.leader.equals(LeaderNode) then
13:            leaderRole ← vertex with LeaderRole
14:            if group.leader.contains(Operator) then
15:                leaderRole.append(OperationalTask)
16:            end if
17:            roleSet.add(leaderRole with
                   ReceivingTask & SendingTask
                   for each member in group.members)
18:        end if
19:        for each member in group.members do
20:            roleSet ←makeRole(member)
21:        end for
22:    end if
23:    return roleSet
24: end function
```

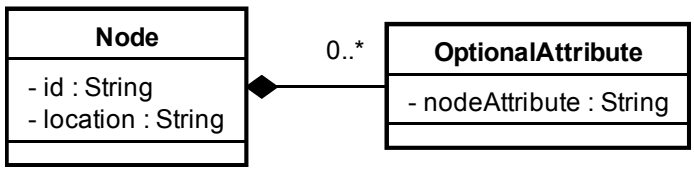


図 3.9: NetInfoML のメタモデル

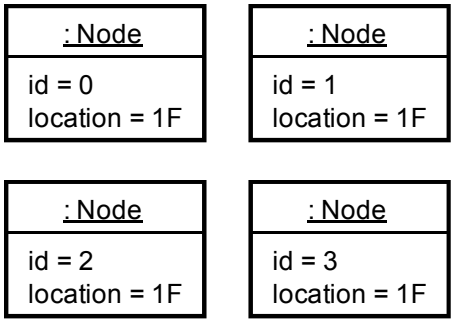


図 3.10: NetInfo モデルの記述例

する処理を，そうでない場合は再度条件分岐を行う．最初の探索対象は Group であるので，入力モデルの最上位要素である Group を取得する (行 9)．当該グループのリーダは Sink であるため，条件分岐により対応する BaseStationRole を受信タスク (ReceivingTask) とともに作成する (行 10-11)．その後，当該グループの各メンバに対して makeRole を呼び出す (行 19-21) ことで，下位のグループに対しても Role の作成を行う．例では中間層に当たる Group 以下を入力として makeRole を呼び出す．リーダに集約処理を持つ LeaderNode，メンバに子グループを持つ Group を入力として再帰的に makeRole を呼び出す．

この場合は条件分岐の結果，LeaderNode に対応する LeaderRole が生成される (行 12-18)．更にメンバ要素を用いて makeRole の再帰呼び出しを繰り返し探索，変換を進める．makeRole において，入力が MemberNodes であった場合には対応する MemberRole を作成する (行 6-7)．これは，Group-level モデルにおいては MemberNodes より下位に接続される要素が存在しないためである．再帰呼び出しによる自動変換の結果，図 3.8 下部に示すモデル中の Role が得られる．

Role の生成後は，Group-level モデルの LeaderNode，MemberNodes が持つ属性 selectionPattern と，Node-level モデルの各 Role が持つ属性 deploymentCondition に基づき，条件を満たす Node に割当を行う．この際，開発者は開発対象とする WSN を構成するノードに関する情報を別途指定する必要が生じる．本フレームワークでは，このノード情報を記述するためのメタモデル (以下，NetInfoML) も定義した．センサノードを表す Node は，頻用される属性として id と location を持つ．また，開発対象ごとに特化した属性をノードに付与することを考え，各 Node が追加の属性 OptionalAttribute を持つよう定義した．図 3.9 に定義したメタモデルを示す．

また，図 3.10 に NetInfoML によるモデル記述例を記す．このモデルは，対象とする WSN が 4 つのノードから構成されていること，それぞれのノードに ID，場所のみが設定

されていることを示している．

本研究ではまず生成された Role からなる Node-level モデルと，開発者により記述された NetInfo モデルを合成し，その後各 Role が持つ属性 deploymentCondition に基づき Node への関連を生成する．現状では単純な割当語彙としており，いずれか一つ以上のノードへの割当を行う “Any”，全てのノードへの割当を行う “All” を対象としている．また，Node-level モデル上の deploymentCondition と NetInfo モデル上の OptionalAttribute の属性 nodeAttribute との文字列が完全一致する場合にも割当を行う．

図 3.8 の例では，BaseStationRole と LeaderRole はいずれか一つの Node に，MemberRole はいずれの Role も割り当てられていないすべての Node にそれぞれ割り当てられている．

このように Group-level から Node-level へのモデル変換では，Group-level モデルが持つ処理を Role とそれに付随する Task として切り出し，個別ノードへの割り当てを自動で行う事により Node-level モデルの自動生成を実現する．

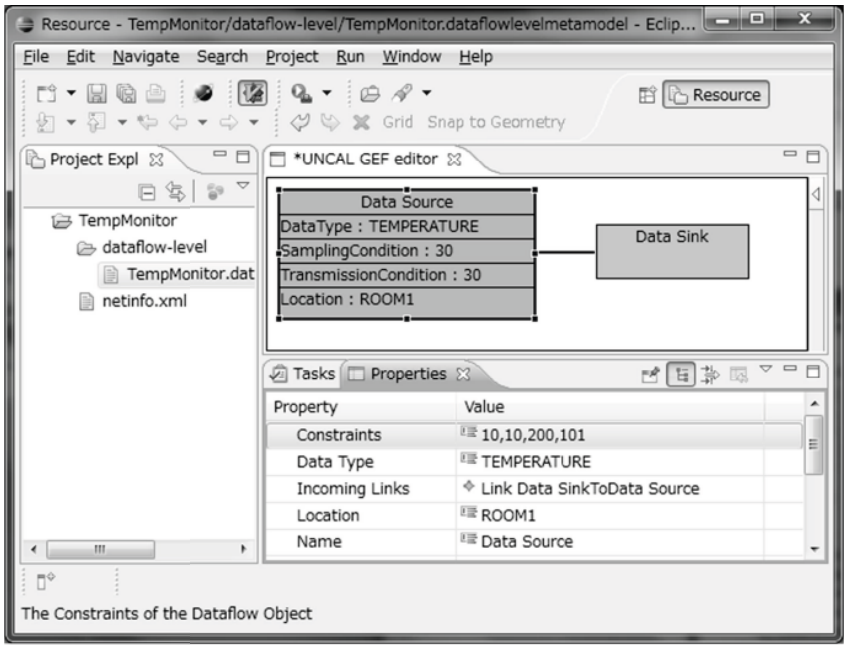
3.3　MDD フレームワークの実装詳細

本節ではこれまで述べたフレームワークのツール実装について詳述する．本ツールは，Java を主体とした統合開発環境である Eclipse [1] のプラグインとして，Eclipse modeling framework(EMF) [2] 上に実装した．ツールは 3.1 節で提案した DSML によるモデルをグラフィカルに編集するためのモデリングツール，3.2 節で提案した PIM 間のモデル変換規則，実装の獲得及びプラットフォーム間の移植のために必要な PIM・PSM 間変換及びコード生成器からなる．

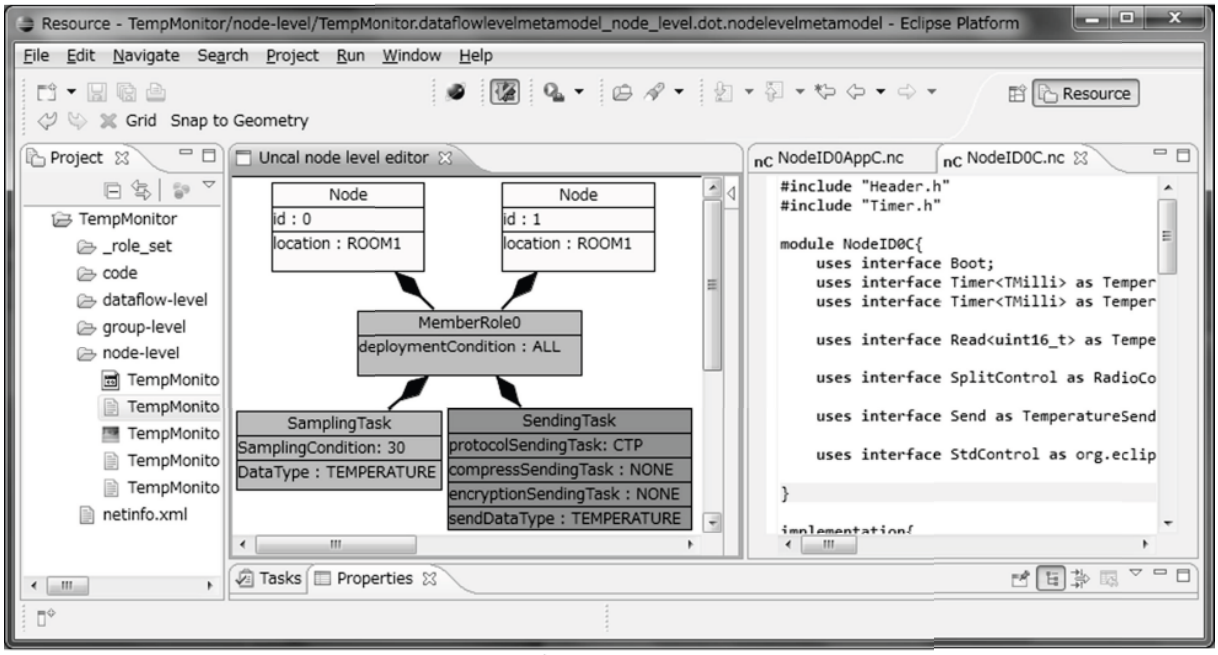
本ツールの動作画面の一部を図 3.11 に示す．図 3.11(a) は DataflowML のためのグラフィカルモデリングツールであり，クラスや属性，リンクの編集を Eclipse 上で実装している．また，図 3.11(b) はモデル変換規則を通し自動生成された Node-level モデルの一部と，このモデルから自動生成されたコードの一部をエディタ上で表示している．各構成要素は図 3.1 に示す様に既存技術を基に実装しており，詳細は後述する．

また，実装獲得及び移植のための実装部分の全体像を図 3.12 に示す．これらを実現するためには PIM から PSM，PSM から実装コードへの詳細化，コードから PSM，PSM から PIM への抽象化が必要となる．特定のプラットフォームに向けたコード生成を実現するためには，PIM から PSM への変換及び PSM からコードへの変換が必要である．また，プラットフォーム間の移植を実現するためには，既存コードからのモデル抽出及び PSM か

[1] http://www.eclipse.org/
[2] http://www.eclipse.org/modeling/emf/



(a) DataflowMLのためのモデリングツール



(b) NodeMLのためのモデリングツールと自動生成コード

図 3.11: 提案フレームワークの Eclipse 上におけるツール実装

らPIMへの抽象化変換が必要である．しかしながら，提案したDSMLはプラットフォームに非依存であるものの，上述の変換はプラットフォームごとに開発しなければならず，一般化は困難である．

そこで本研究では，2つの具体的なプラットフォームに対する実装のみについて記す．まず，本研究の対象であるNode-levelプラットフォームに属し，最も頻用されるTinyOSを対象としたPIM2PSM変換(図 3.12 ND2NesC Trans.)と，TinyOS上で動作するNesCコードの生成(図 3.12 NesC Code Gen.)について詳述する．続いて異なる抽象度間の移植を実現するため，Dataflow-levelプラットフォームに属するTikiriDBを対象に，TikiriDBのPSMから同抽象度のPIMに相当するDataflowMLへの変換(図 3.12 TikiriDB2DF Trans.)について詳述する．ただし，本論文ではDSLで記述されたコードからのPSM抽出部については対象外としているが，この部分についてはEMFを用いたDSL開発支援技術 [32]や，EMF上で実装されたモデル獲得技術 [15]により補完可能である．

Graphical model editors

モデリングツール部は，Eclipse上でのモデルエディタ実装用フレームワークであるGraphical Editing Framework(GEF) [3]を用いた．DataflowML，GroupML，NodeMLの文法はメタモデルとしてEMF上で定義し，このメタモデルに基づきモデリングツールを実装した．記述されたモデルは全てXML Metadata Interchange (XMI)形式で表現されるため，のちのモデル変換やコード生成にてシームレスに利用可能である．

PIM2PIM transformations

PIM間のモデル変換部は，モデル変換言語として広く用いられているATLAS Transformation Language(ATL) [4]を用い，前述の定義に従い実装した．ATLを用い，各DSMLのメタモデル上での対応関係や追加情報を実装することで，Dataflow-levelからGroup-level，Group-levelからNode-levelへの自動モデル変換を実現している．本実装では`Role`の`Node`への割当部分もATLを用いて実装した．

[3] http://www.eclipse.org/gef/
[4] http://www.eclipse.org/atl/

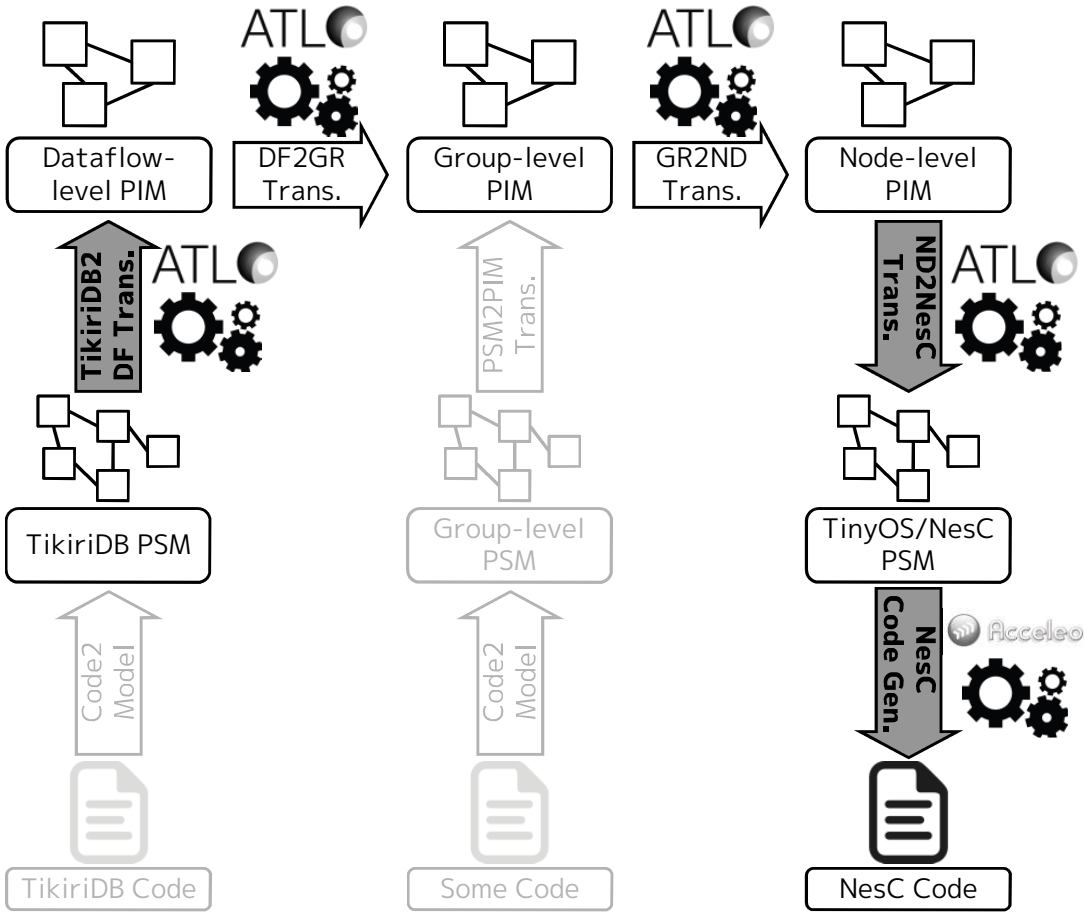


図 3.12: 提案フレームワークにおける PIM・PSM 変換部とコード生成部

PIM2PSM transformation

PIM から PSM への変換部は ATL を用い，NesC を対象に実装した．ここでは PIM2PSM 変換の概要について述べる．

NesC の PSM については既存の PSM 定義 [75] を用いている．Node-level PIM から NesC PSM への変換では，主に Node-level PIM 内の各 `Task` を，その動作を実現する NesC コンポーネントに対応付けることで実現している．図 3.13 にこの変換例の一部を示す．ここでは，Node-level PIM における `SamplingTask` が，計測動作を定期的に実行するための NesC コンポーネントである `Read` と `Timer` に変換されている．このような NesC コンポーネントへの変換規則を全ての `Task` に対して定義することで PIM2PSM 変換を実現した．我々の変換では NesC コンポーネント郡を各 `Node` とそれに紐づく `Role` と `Task` から生成している．すなわち，実装をノードごとに生成することを意図している．

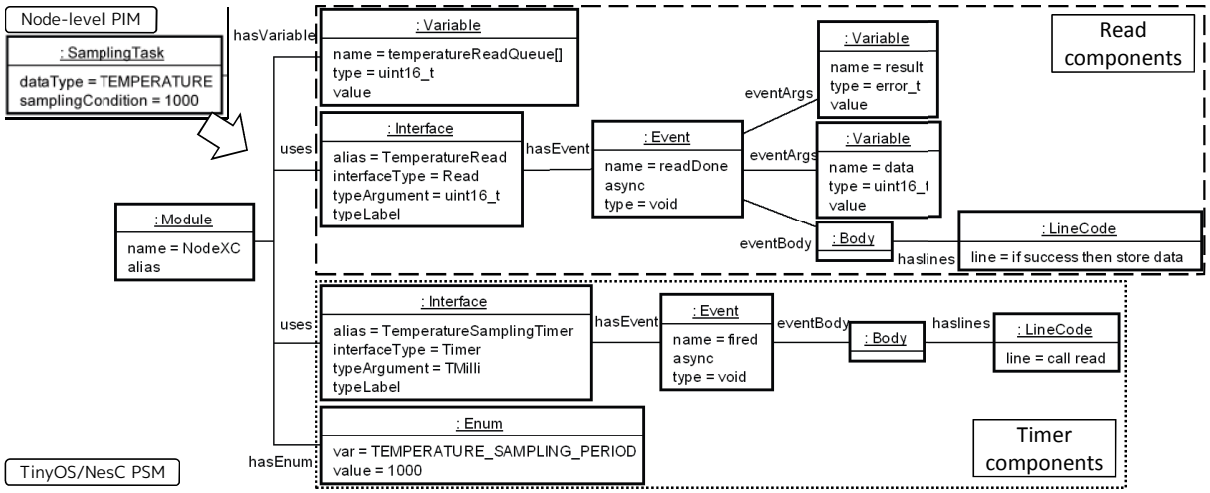


図 3.13: Node-level PIM から NesC PSM への変換例の一部

Code generator

コード生成部の実装にはテンプレートベースのコード生成フレームワークである Acceleo [5] を用いた．Acceleo は Object Management Group が提供する model to text 変換言語の仕様の実装を提供しており，モデル駆動開発にかかる実装にて頻用されている．本研究では対象プログラミング言語を NesC とし，NesC PSM を入力とし NesC コードを出力するようテンプレートを実装した．

今回用いた NesC PSM は，NesC のシンタックスと非常に近しいため，コード生成はほぼ一対一の対応で実現可能である．NesC PSM には，具体的なコードやコード片の情報は含まれていない．したがって，NesC におけるコード生成のためのテンプレートは，図 3.14 に示す様にコード断片を主としたものとなる．例えば，図 3.14 上部に示す NesC のテンプレートは，データ計測が成功した際に，得られたデータを格納するコードの生成に用いる (lines 8–13)．このテンプレートにより自動生成されたコード例は図 3.14 下部に示すものである．コード生成では，アプリケーション固有の処理部分はスケルトンコードとして生成し，後に人手での実装の組み込みを必要とする．

現状のコード生成では，TinyOS に組み込まれているセンサの計測機能を提供するコンポーネントの利用を前提としている．現状の TinyOS では輝度，温度，湿度，加速度といった離散的に計測する物理情報を取得するセンサがサポートされており [6]，本研究もこれらを対象としている．その他，マルチメディアなどの複雑なデータを扱う場合には，本

[5] http://www.eclipse.org/acceleo/
[6] https://github.com/tinyos/tinyos-main/tree/master/tos/sensorboards

```
// Used Events Declaration
[for (eve : Event | uses.hasEvent)]
event [type.toLower()/] [name/]([for (arg : Variable | eventArgs)][if (i > 1)], [/if][type/] [name/][/for]){
    [for (lc : LineCode | eventBody.hasLines)]
        [if (line = 'boot')]
    ...
        [elseif (line = 'read_done')]
            [let data : String = eve.name.substring(1, eve.name.index('Read')-1)]
    if(result == SUCCESS){
        [data.toLower()/]ReadQueue['['/][data.toLower()/]ReadPointer[']'/] = data;
        [data.toLower()/]ReadPointer = ([data.toLower()/]ReadPointer + 1) % [data.toUpper()/]_READ_QUEUE_LEN;
    }[/let]
        ...
        [/if]
    [/for]
}[/for]
```

```
event void TemperatureRead.readDone(error_t result, uint16_t data){
    if(result == SUCCESS){
        temperatureReadQueue[temperatureReadPointer] = data;
        temperatureReadPointer = (temperatureReadPointer + 1) % TEMPERATURE_READ_QUEUE_LEN;
    }
}
```

図 3.14: NesC コード生成のためのテンプレートと生成されたコード例の一部

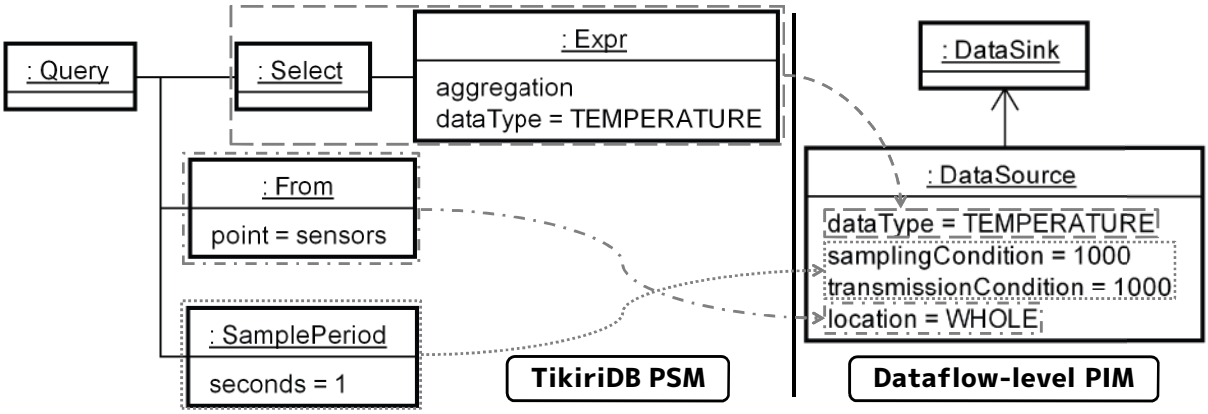


図 3.15: TikiriDB PSM から Dataflow-level PIM への変換例

フレームワークでは個別の実装とコード生成用テンプレートを用意する必要がある．

PSM2PIM transformation

PSM から PIM へのモデル変換もこれまでと同様 ATL を用いて実装した．対象は TikiriDB とし，PSM の定義は TikiriDB のクエリシンタックスに基づきメタモデルを定義した．TikiriDB は Dataflow-level に属する PIM であるため，この PSM から Dataflow-level PIM への変換を本フレームワークでは実装している．

TikiriDB PSM と Dataflow-level PIM は，いずれも WSN ソフトウェアが扱うデータ処理設計のみを扱っている．従って PSM2PIM 変換はほぼ 1 対 1 対応で実現した．変換規則では，TikiriDB PSM 上の `SELECT` 節に含まれる `Expr` ごとに，Dataflow-level PIM の `DataSource` を生成する．この際，`Expr` が集約処理を含む場合は同時に `AggregationPoint` も生成する．TikiriDB PSM 上の `FROM` 節，`SAMPLE PERIOD` 節はそれぞれ `DataSource` の属性に対応付けるよう変換規則を実装した．この PSM2PIM 変換の例を図 3.15 に示す．

第4章　WSNソフトウェア新規開発のための多段階モデル駆動開発プロセス

4.1　はじめに

本章ではWSNソフトウェアの新規開発支援に向けた，多段階MDDフレームワークを用いた開発プロセスを提案する．2.5節で述べた様に，既存のWSNソフトウェア開発のためのMDD手法では，開発対象ごとの品質改善のための要件を満たすことができない．新規開発において求められる要件を以下に再掲する．

- 初期実装獲得工程における要件

  要件1. 新規開発における低コストでのプロトタイプ開発が可能であること

- 品質改善工程における要件

  要件3. 通信設計，タスク割当設計を含む細やかな設計変更が可能であること

  要件4. 品質改善のための設計変更を端的に記述可能であること

しかしながら，抽象度の高いDSMLを用いるMDDでは，その抽象化のためモデル上での記述が限定的であり要件3を達成できない．また抽象度の低いDSMLを用いるMDDでは，関心事の混在によりモデル上での記述が複雑となり，要件1，4を達成できない．すなわち，単一の抽象度のみを扱う既存研究では，抽象度間のトレードオフのため，上述の要件を全て同時に満たすことができない．

そこで第3章では複数の抽象度を併用するというアイデアを実現するため，3種のDSMLと，それらを併用するためのフレームワークを提案した．本章はこのフレームワークの新規開発支援への適用について論じるものである．提案手法として，4.2節にて工程ごとに異なる抽象度のDSMLを使い分ける開発プロセスを定義する．提案したプロセスの有用

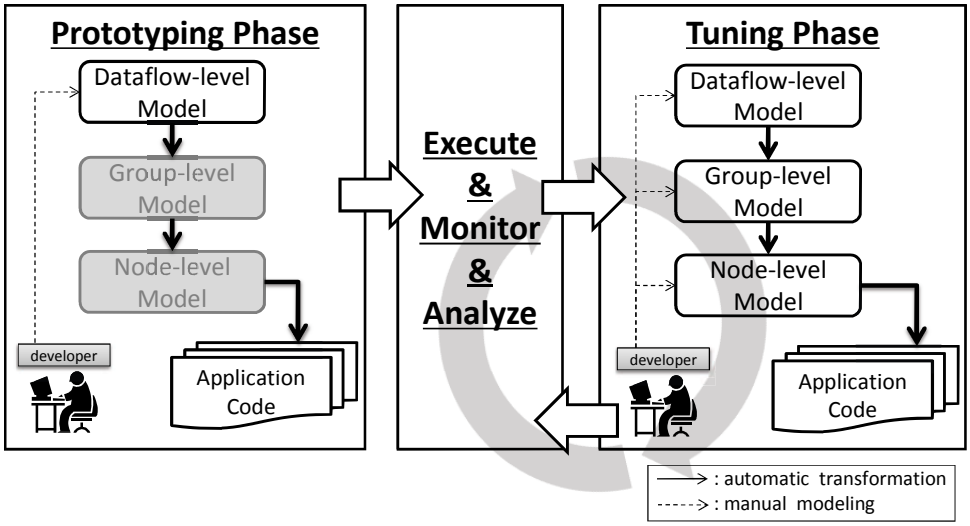


図 4.1: 新規開発のための開発プロセスの全体像

性を評価するため，既存の WSN ソフトウェアを対象としたケーススタディとユーザ試験を実施した．この結果をそれぞれ 4.3 節，4.4 節に示し，4.5 節にて本章をまとめる．

4.2　WSN ソフトウェア新規開発のための開発プロセス

図 4.1 に提案する開発プロセスの全体像を示す．本開発プロセスでは，プロトタイプ開発工程 (Prototyping Phase) ではデータ処理設計のみを扱い，品質改善 (Tuning Phase) ではデータ処理設計，通信設計，タスク割当設計の全てを扱う．

プロトタイプ開発工程では，開発者は対象とする環境で満たすべき機能要求に当たるデータフローを，DataflowML を用いてモデル化する．すなわち，この工程ではデータ処理についての設計のみに焦点を当てており，通信，タスク割当の処理についての設計は扱わない．

開発者によるデータフロー設計後，フレームワークは記述された Dataflow-level モデルを入力とするモデル変換により Group-level モデル及び Node-level モデルを自動生成する．Group-level モデルはデータ処理を実現するノード集合のためのタスクアーキテクチャを表現しており，データ処理についてだけでなく通信設計も含んでいる．すなわち，Dataflow-level モデルはネットワーク非依存であるのに対し，Group-level モデルはネットワーク依存である．また Node-level モデルは Group-level におけるタスクをノードに割り当てることで個々のノードの動作を表現しており，タスク割当設計も含んでいる．開発者

は通信に関わる問題を Group-level モデル上で扱い，タスク割当に関わる問題を Node-level モデル上で扱う．

最後に，フレームワークは Node-level モデルを入力とし，特定のプラットフォームに向けた PSM，プラットフォーム上で動作するプログラムを自動生成する．データフロー設計のみを扱い，モデル変換，コード生成を自動化することで，“要件 1. 低コストでのプロトタイプ開発” を達成する．

品質改善工程では，開発者は非機能要求を満たすよう，任意の抽象度の DSML を用いてモデルを記述する．この際，開発者は Group-level や Node-level のモデルをスクラッチから記述せず，自動生成されたモデルに対する修正を行う．

Group-level モデル上ではルーティングプロトコルや圧縮アルゴリズムの選択が可能である．これらの設計の選択は，WSN 専門家が用意する，既存の再利用可能な設計と対応する語彙を予め用意し，その語彙の中から利用したい設計に対応する語彙を Group-level モデル上で明示することで実現される．例えば，開発者は性能にトレードオフを持つ既存のルーティングプロトコル [58, 70] の中から，非機能要求に応じていずれかを選択しモデル上に反映する．

Node-level モデル上ではタスク割当に関する設計が可能である．Node-level モデルは対象とする WSN を構成するノードと，要求を達成するためのタスクから構成されるため，どのノードにどのタスクを割り当てるかを最適化できる．タスク割当の設計変更は，モデル上ではノードとタスクの関連の変更として反映可能である．これら複数の抽象度の DSML の併用により，“要件 3. 細やかな設計変更” と “要件 4. 端的な設計変更の記述” を両立する．

以下では病院内での患者監視に WSN を適用したシナリオ [21] を用いて本開発プロセスの各工程における具体例を示す．患者監視ソフトウェアでは患者の状態をリアルタイムに把握するため，患者に取り付けたセンサから得た心拍数，血中酸素濃度のデータを収集する．この患者監視ソフトウェアは，アメリカ合衆国ミズーリ州にある Barnes-Jewish 病院にて 60 個のノードを 41 日間に渡り配備した実績を持つ．

4.2.1 プロトタイプ開発工程

本工程では，開発者はセンサデータの処理に関する設計を行い，その結果を，DataflowML を用いて機能要求を満たす Dataflow-level モデルをして記述する．患者監視における機能要求は，心拍数 (PULSE) と血中酸素濃度 (OXYGEN) の 2 種類のセンサデータを計測，送信，収集することである．図 4.2 にこの機能要求を満たすための Dataflow-level モデル例を示す．ここでは 2 つの `DataSource` がそれぞれ 2 種類のデータを計測，送信し，

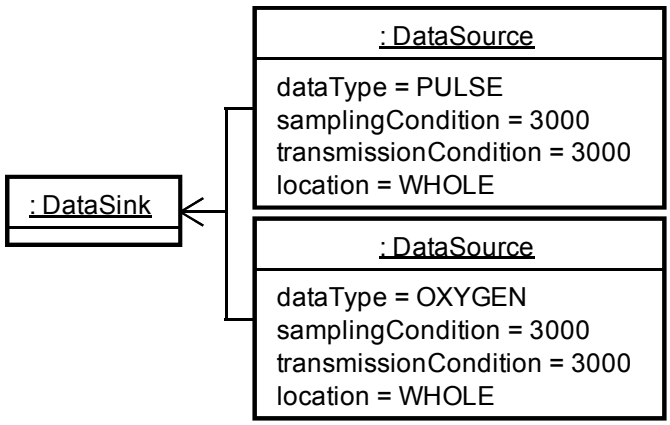


図 4.2: 患者監視における Dataflow-level モデル例

リンクで接続された DataSink がこれらを受け取ることそれぞれ表している．

Dataflow-level モデル記述後は，記述されたモデルを入力とし、フレームワークから詳細化された Group-level モデル，Node-level モデルと，これらに対応する実装が出力される．これによりプロトタイプ開発を達成する．次節では生成された実装を実行し，品質を測定，分析した結果，品質改善のための設計変更を行う品質改善工程について記す．

4.2.2　品質改善工程

本工程では，品質改善のための設計変更を，任意の抽象度を用い，自動生成されたモデルに対する修正することで実施する．以下では Group-level，Node-level における設計変更について述べる．

Group-level モデル上では，開発者はノード集合に対して割り当てられるタスクのアーキテクチャと，タスク間での通信方式を設計する．GroupML を用いることで，開発者は WSN 内の通信を，タスク間の通信というマクロな視点から記述可能となる．

図 4.3 に，患者監視における自動生成された Group-level モデル例と，品質改善のための設計変更の様子を示す．このモデル上では階層的なグループクラスタリングが示されている．これは暗示的にデータフローを表しており，子に当たる Group がセンサデータを計測，送信し，親に当たる Group が下位 Group からデータを収集している．また，このモデル上ではルーティングプロトコルとしてツリー型のトポロジを用いること (topology=TREE)，データ圧縮や暗号化は行わないこと (compression=NONE, encryption=NONE) がそれぞれ対応する属性とその値として表現されている．

この自動生成されたモデルに対し，開発者は必要に応じて品質改善のための通信設計変

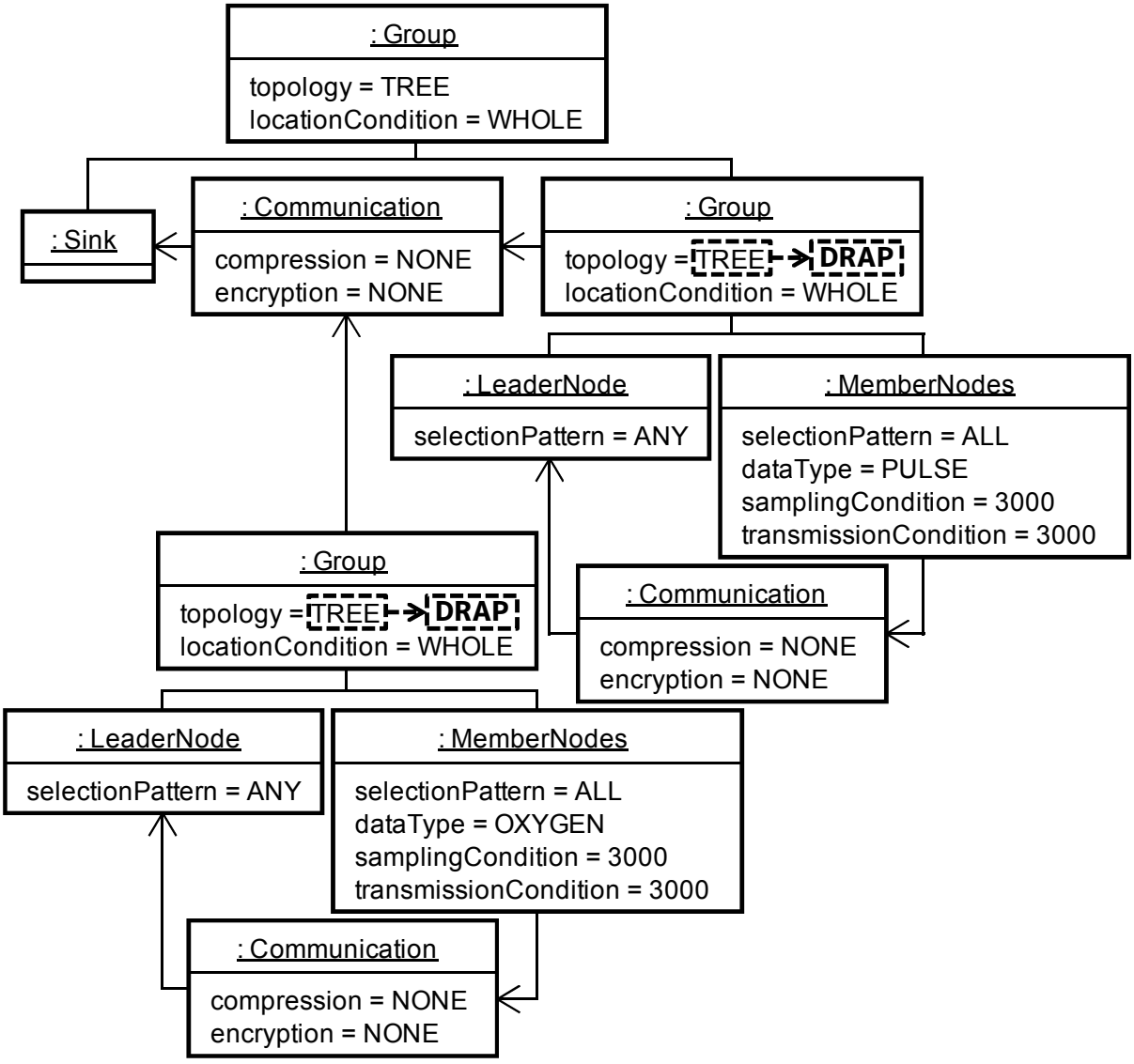


図 4.3: 患者監視における Group-level モデル例と通信設計変更例

更を行う．例えば，Chipara らは DRAP(Dynamic Relay Association Protocol) と呼ぶ独自のプロトコルを採用している．この様なルーティングプロトコルの変更は，図 4.3 に示すように，計測に関わる `Group` における属性 `topology` の値を対応する語彙である DRAP に変更することで達成可能である．また，通信設計変更の後は，使用する語彙に対応するテンプレート実装を用意することで通信設計を反映するコードを自動生成可能である．

図 4.4 に，患者監視における Node-level モデル例から，心拍数計測にかかるタスクのみを抽出したものを示す．図 4.4“Before” のモデルでは，データの計測，送信タスクと，これらのデータを中継するタスク，すべてのデータを収集するタスクが 7 つのノードに対し割り当てられていることが示されている．データ収集タスクはベースステーションとなる ID 0 のノードが，中継タスクは適当に選択された一つのノードが，計測と送信のタスクはその他全てのノードがそれぞれ担っている．

このモデルに対し，開発者は Group-level モデルの場合と同様，品質改善のための設計変更を適用する．例えば，対象とする環境が広域である場合，通信の信頼性，すなわちデータ到達率を向上させるために，図 4.4“After” のモデルに示す様に，より多くのノードに中継タスクを割り当てるとった設計変更が考えられる．提案フレームワークでは各ノードに対してコード生成を行うため，タスク割当の変更は生成されるコードの違いとして実装に反映される．

4.3　ケーススタディ

本開発プロセスの適用可能性を確認するため，文献にて報告されている，実世界での運用実績を持つ 5 つのアプリケーションを開発するケーススタディを実施した．本ケーススタディに対応する 3 つの研究課題を以下に示す．

RQ.1 単一の抽象度を用いた場合と比較し，プロトタイプ開発を少ないモデル記述コストで実現できるか

RQ.2 現実的な WSN ソフトウェア品質改善のための設計変更を，本開発プロセスにて記述できるか

RQ.3 単一抽象度を用いた場合と比較し，品質改善のための設計変更をより簡潔に記述できるか

RQ1 は要件 “新規開発における低コストでのプロトタイプ開発が可能であること” に対応する．RQ2 は要件 “通信設計，タスク割当設計を含む細やかな設計変更が可能であるこ

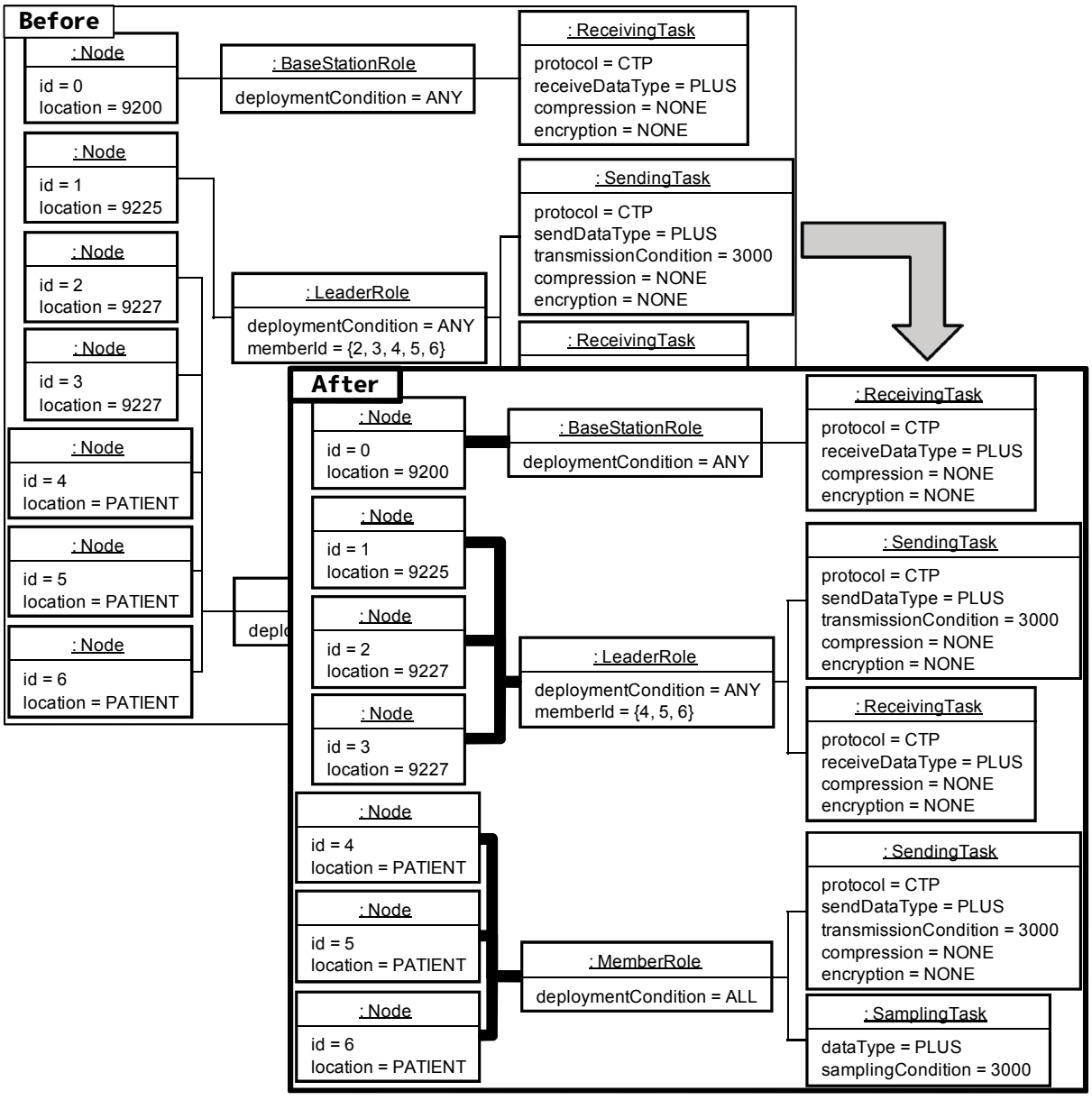


図 4.4: 患者監視における Node-level モデル例の一部とタスク割当設計変更例

表 4.1: ケーススタディで用いるアプリケーションのまとめ

アプリ名	データ処理設計		通信設計及び
	対象データ	データ処理	タスク割当設計
歴史的建造物監視 [18]	加速度，歪み，温度，湿度	Bayesian アルゴリズム，時系列平均	ハフマン圧縮，設置場所によるタスク割当
患者状態監視 [21]	心拍数，血中酸素濃度	なし	ルーティングプロトコル、役割によるタスク割当
火災監視 [27]	温度，湿度，気圧	なし	なし
気候監視 [40]	温度，湿度，風向，風速	なし	なし
橋梁監視 [46]	加速度	時系列平均	ルーティングプロトコル

と” に，RQ3 は要件 “品質改善のための設計変更を端的に記述可能であること” にそれぞれ対応する．

4.3.1　ケーススタディの設計

表 4.1 にケーススタディで用いたアプリケーションの概要を示す．“データ処理設計” の列はアプリケーションに必要なデータの種類と処理を，“通信設計及びタスク割当設計” の列は品質改善のために実施された設計変更をそれぞれ示す．対象ソフトウェアは，開発成果が文献で報告されているものの中から，WSN ソフトウェア調査論文 [8] [62] を参考に，2.1 節にて述べた，最も典型的な WSN ソフトウェアに該当するものを選択した．これは，本研究で用いる DSML が，定期的な計測・通信を行う最も典型的な WSN ソフトウェアを対象としているためである．

ケーススタディでは上記の WSN ソフトウェアを対象に，プロトタイプ開発工程，品質改善工程におけるモデルの記述コストの測定及び記述可能な設計の検証をすることで研究課題への回答を試みる．本ケーススタディでは，プロトタイプ開発工程においては開発にかかるコストを，品質改善工程においては決定された設計変更を反映するモデル修正にかかるコストをそれぞれ調査する．しかしながら，開発コストを直接，正確に測ることは困難であることが知られている [13]．したがって本ケーススタディでは，開発コストのひ

とつの指標として頻用されるソフトウェアの大きさ，すなわち記述したモデルサイズを測定する．また，モデル修正コストは修正前後のモデル上での編集距離として考えられる．本ケーススタディでは，これら 2 つを同時に扱うため，モデリングステップと呼ぶ尺度を導入する．これは，モデル上のクラス，リンク，属性に対する追加，削除，変更の各操作を 1 ステップとし，モデルの作成や修正にかかる操作回数の合計を測るものである．データ収集のためのモデル作成及び修正の流れは以下に記す．なお，各ケーススタディでは，モデルの作成及び修正は著者らで実施した．

モデリングステップの測定手順:

- プロトタイプ開発工程

  (1) 各ソフトウェアの機能性を，文献を参照し同定，機能性を満たすモデルを記述

    (1.1) (多段階 MDD のみ) 自動モデル変換を通し Group-level モデル，Node-level モデルを生成，取得

- 品質改善工程

  (2) 各ソフトウェアの品質改善のための設計を，文献を参照し同定

  (3) 通信方式にかかる設計を同定し，設計を反映するよう既存のモデルを修正

    (3.1) (多段階 MDD のみ) 修正済みモデルを入力とし，自動モデル変換器を通し修正が反映された下位のモデルを生成，取得

  (4) タスク割当にかかる設計を同定し，設計を反映するよう既存のモデルを修正

    (4.1) (多段階 MDD のみ) 修正済みモデルを入力とし，自動モデル変換器を通し修正が反映された下位のモデルを生成，取得

上記の手順は，2 種類の MDD プロセスに対して実施，比較する．本研究にて提案する多段階 MDD プロセス (P_{multi}) では 3 つの DSML を併用する．プロトタイプ開発工程では DataflowML を，品質改善工程では 3 つの DSML を使い分けてモデルの記述と修正を行う．比較対象は，Thang ら [91] の様に開発中に Node-level に属する DSML のみを用いるプロセス (P_{uni}) とし，本ケーススタディでは NodeML のみを用いる．すなわち，プロトタイプ開発工程，品質改善工程のいずれも NodeML にてモデルの記述と修正を行う．DataflowML または GroupML のみを用いる場合は，通信，タスク割当のいずれか或いは双方がモデル上記述できず，設計上の柔軟性が失われ細やかな品質調整を達成できない．したがって本ケーススタディでは NodeML のみを比較対象とした．各々の開発プロセスにおけるモデルの記述と修正の作業に対してモデリングステップを測定する．

表 4.2: P_{multi} と P_{uni} におけるモデリングステップ (データフロー設計)

アプリ名	$\boldsymbol{P_{multi}}$			$\boldsymbol{P_{uni}}$
	DataflowML	**NetInfoML**	合計	
歴史的建造物監視	38	49	87	232
患者監視	15	124	139	353
火災監視	19	31	50	165
気候監視	13	19	32	101
橋梁監視	15	193	208	299

表 4.3: P_{multi} と P_{uni} におけるモデリングステップ (通信設計)

アプリ名	$\boldsymbol{P_{multi}}$	$\boldsymbol{P_{uni}}$
歴史的建造物監視	2 (GroupML)	4
患者監視	2 (GroupML)	4
火災監視	変更なし	変更なし
気候監視	変更なし	変更なし
橋梁監視	2 (GroupML)	4

収集した P_{multi}，P_{uni} それぞれのモデリングステップ数 $ModelingStep_{multi}(MS_{multi})$, $ModelingStep_{uni}(MS_{uni})$ を，ケーススタディごとに比較する．これにより，各プロセスにて開発にかかるモデル記述コストの比較を行う．表 4.2〜 4.5 は各 WSN ソフトウェアの開発と設計変更にかかるモデリングステップ数の一覧を示している．表 4.2 は前述の測定手順 1 にかかるモデリングステップを示している．同様に表 4.3 は測定手順 3，表 4.4 は測定手順 4 のそれぞれにかかるモデリングステップを示している．これらの値を P_{multi}, P_{uni} ごとに合計した結果を，表 4.5 に示している．

4.3.2　Case Study 1: 歴史的建造物監視

歴史的建造物監視 [18] は 2.1 節で示した様に建築物の異常状態を検知するための監視を行う．当該文献では 4 種のデータとベイジアンアルゴリズムを用い，イタリアにある中世に建築された塔を対象とした配備を行ったことが報告されている．歪みのデータはセンサの特性上非常に不安定であるため，過去 10 回の計測の平均値を代表値として送信している．配備は 3 階建ての塔に対し，16 個のノードを用い，4ヶ月に渡る監視を実施した実績を持つ．

表 4.4: P_{multi} と P_{uni} におけるモデリングステップ (タスク割当設計)

アプリ名	$\boldsymbol{P_{multi}}$			$\boldsymbol{P_{uni}}$
	DSML	**NetInfoML**	合計	
歴史的建造物監視	89 (NodeML)	0	89	89
患者監視	4 (GroupML)	41	45	58
火災監視	N/A	N/A	変更なし	変更なし
気候監視	N/A	N/A	変更なし	変更なし
橋梁監視	N/A	N/A	変更なし	変更なし

表 4.5: P_{multi} と P_{uni} における総合モデリングステップ

アプリ名	$\boldsymbol{P_{multi}}$	$\boldsymbol{P_{uni}}$
	$\boldsymbol{MS_{multi}}$	$\boldsymbol{MS_{uni}}$
歴史的建造物監視	178	325
患者監視	186	415
火災監視	50	165
気候監視	32	101
橋梁監視	210	303

品質改善にあたってはハードウェア，配備方式，OS，ソフトウェアなど様々な側面を考慮している．ソフトウェア部については，加速度データに対するハフマンデータ圧縮を適用し通信量の低減を実施している．これは，加速度データは他のセンサデータより高頻度で計測されるため送信するデータ量が増加し，WSN の生存期間が短くなってしまうことに対する設計変更である．また，タスク割当については，建築学の知識に基づきセンサノードの配置場所に基づき実施している．つまり，Ceriotti らは，タスク割当をノードの物理的な配置場所によって決定している．これにより最終的に得る異常判定のデータ精度という品質向上を計っている．

プロトタイプ開発工程

本開発プロセスを用いる場合，HBM のデータ処理設計は DataflowML を用いて図 4.5 に示すモデルとして記述可能であった．HBM は，加速度，歪み，温度，湿度のデータを計測し，集めたデータから建築物の異常を検知する．この際，歪みデータのみ過去の計測値の平均値を送信する．図 4.5 に示すモデルは，4 つの `DataSource` が必要な 4 種のデー

タ計測を表しており，FusionPoint が 4 種のデータを用いた建築物の異常検知処理を表す．TemporalAggregationPoint は歪みデータのみ過去の計測値の平均を計算することを表している．このモデルは，アプリケーションが必要とするデータの計測と処理から設計可能である．センサデータの計測には，計測間隔や場所に関する決定も含まれ，これらは DataSource の属性として記述可能である．

品質改善工程

通信に関する設計変更は，加速度データのみ圧縮を行うことで WSN の生存期間の伸長を試みるものであった．この設計変更は，本開発プロセスでは GroupML にて実現可能であった．図 4.6 はモデル変換により自動生成された Group-level モデルの一部と，圧縮アルゴリズムの採用に対応するモデル修正を示している．自動生成されたモデルでは圧縮を用いないという設計が選択されている (compression=NONE)．圧縮アルゴリズムを適用するためには，事前にアルゴリズムの名前と対応するコードテンプレートを用意する．この場合，利用可能な語彙として HUFFMAN が存在するという仮定の元，開発者はモデルの修正を実施する．具体的には図 4.6 に示すよう，加速度の計測を担う Group に接続される Communication が持つ属性 compression の値を NONE から HUFFMAN へと変更する．

この設計変更はモデル変換より下位の Node-level モデルへと伝播する．すなわち，Node-level モデルにおいて加速度を扱う Sending/ReceivingTask が持つ属性 compression の値は全て HUFFMAN となる．最終的にこの設計変更はコード生成により実装まで反映される．

続いて，最終的に得られる建築物の異常判定のデータ精度を向上させるためのタスク割当は，本開発プロセスでは Node-level モデルにおける Node と Role との割当関係の変更として記述可能であった．この割当関係の変更の様子を図 4.7 に示す．

図 4.7 は 3 つのノードに対するタスク割当の再設計を表現しており，“Before” モデルはプロトタイプ開発時に自動生成されたモデル，“After” モデルは自動生成モデルに対し手動で変更を加えたモデルである．今回の自動生成された，図 4.7 中の “Before” モデルでは，変換時に全てのノードに全ての役割を割り当てている．これを，手動変更後の，図 4.7 中の “After” モデルの様に，Node と Role との間の関連を変更することにより，[18] で実施されたノードごとに異なるタスク割当を，NodeML を用いて設計，記述可能であった．

モデリングステップ数

続いてモデルの記述にかかるコストを，モデリングステップを用いて計測し，P_{uni} の場合と比較評価した結果について記す．表 4.2〜 4.5 は，各プロセスにおける，データフ

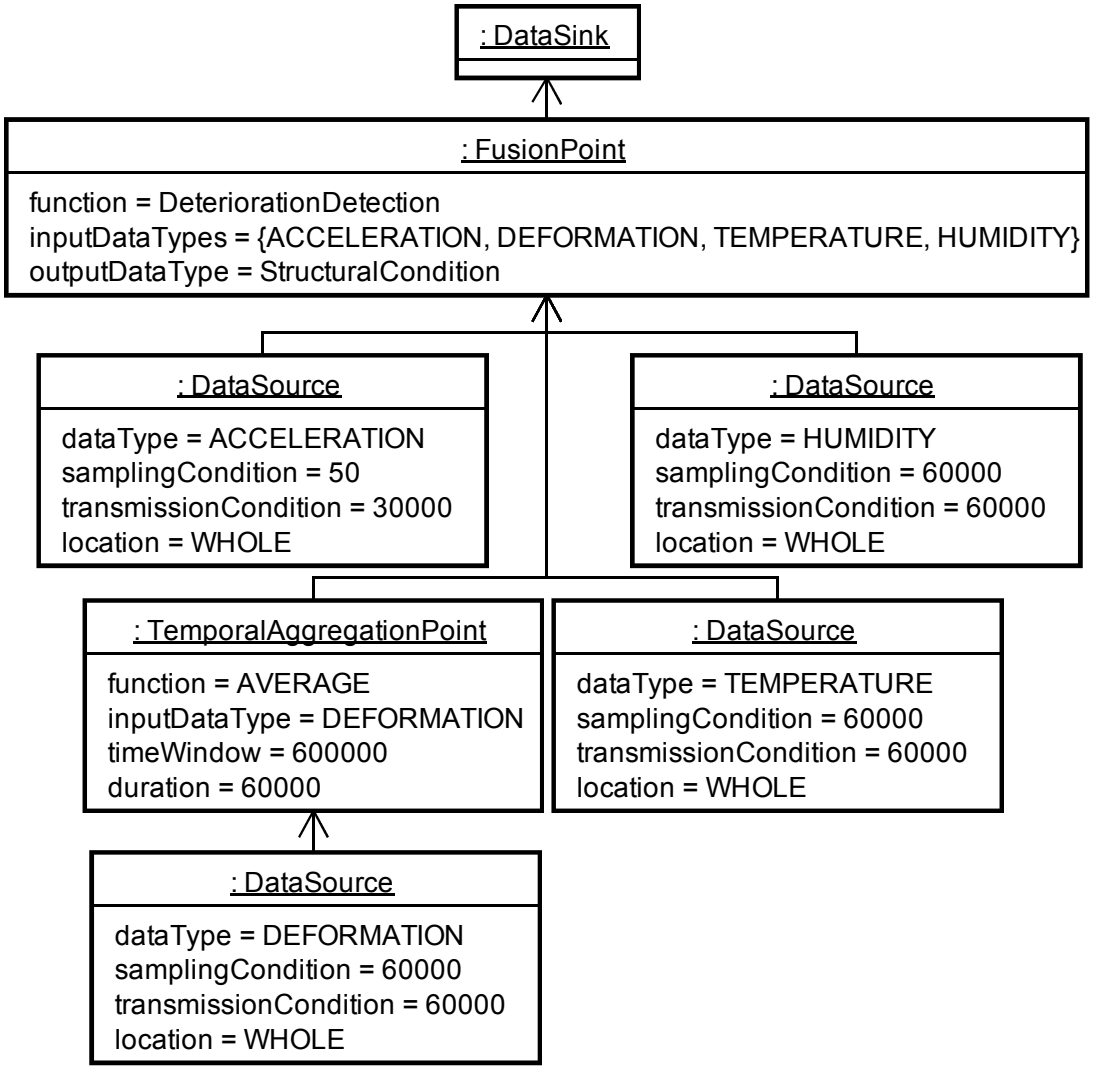


図 4.5: 歴史的建造物監視の DataflowML によるモデル例

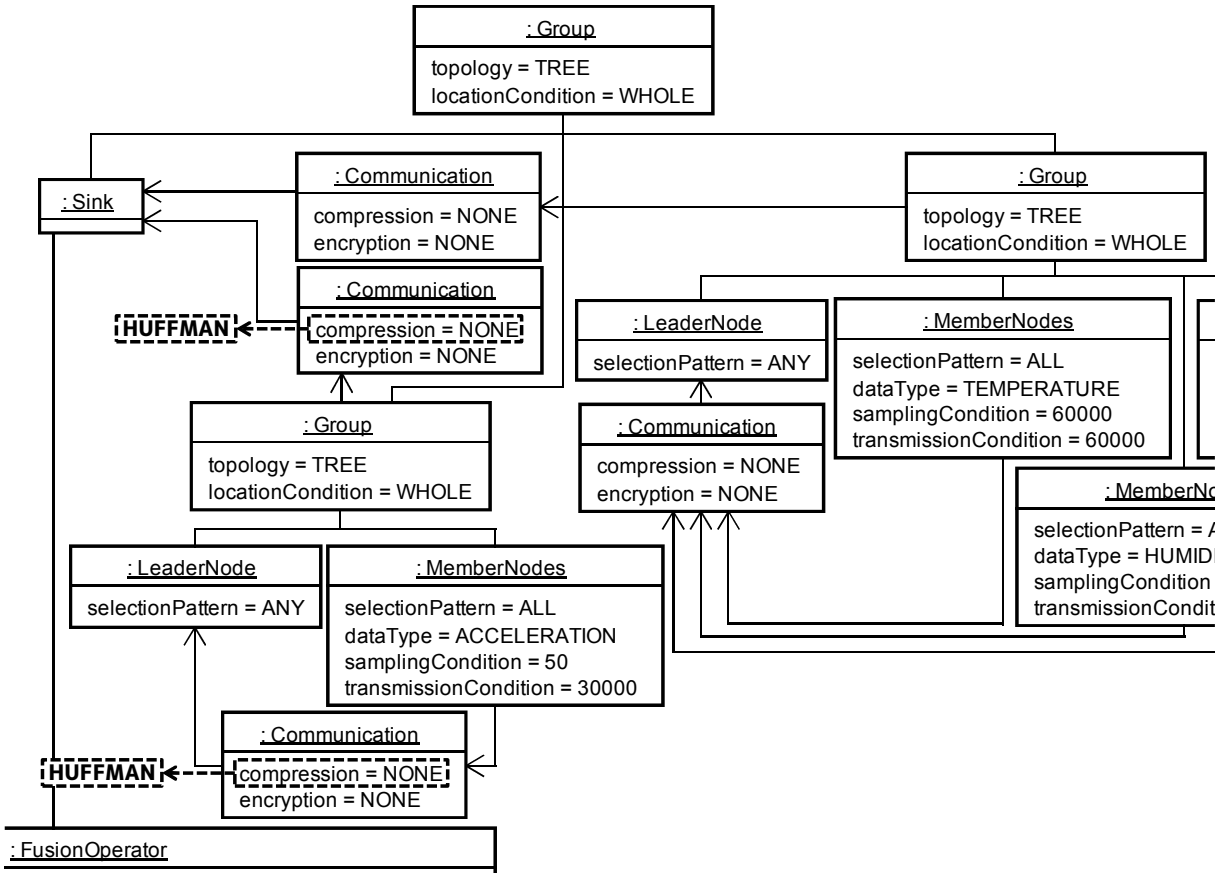


図 4.6: 歴史的建造物監視の GroupML によるモデル例の一部と設計変更例

ロー設計，通信設計，タスク割当設をモデルに反映する際にかかるモデリングステップ数と，それら全てを計測した結果である．

まずデータフロー設計について，P_{multi} では DataflowML によるモデル記述には 38 ステップを要した．これに加えて，NetInfoML による配備先 WSN のノード情報は別途必要であり，これは 49 ステップで宣言可能であった．一方 P_{uni} では，NodeML を用い機能性を実現するための `Task` と `Role`，WSN を構成する `Node` とノードへのタスク割当を記述する必要があり，全体で 232 ステップを要した．

通信設計については，P_{multi} ではデータ圧縮に関わる設計を，GroupML を用いて変更可能であったこの修正は自動生成された Group-level モデルに対して行い，2 ステップでの修正を達成した．一方 P_{uni} では，この修正は加速度データの送信・受信を担うタスク (`Sending/ReceivingTask`) の 2 箇所の属性 `compression` に分散するため 4 ステップを要した．

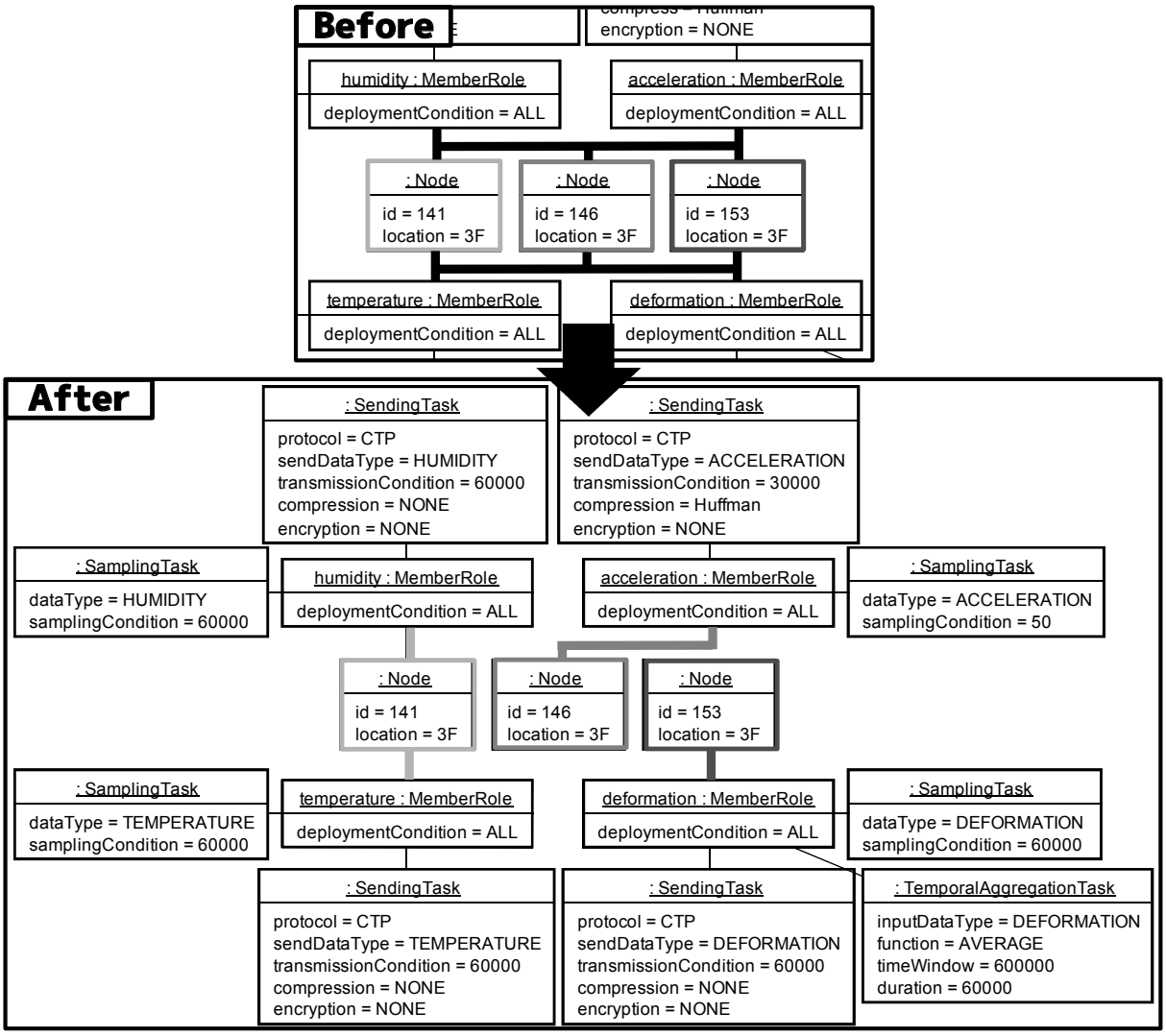


図 4.7: 歴史的建造物監視の NodeML によるモデル例の一部と設計変更例

タスク割当設計について，状態異常判定の精度向上のためのタスク割当は，P_{multi}，P_{uni} ともに NodeML を用いた修正で実現し 89 ステップを要した．

以上の結果をまとめると，表 4.5 より，データフロー設計からタスク割当設計まで同時に考慮する $ModelingStep_{uni}$ は 325 ステップであったのに対し，設計対象ごとに抽象度を選択可能であった $ModelingStep_{multi}$ は 178 ステップであった．すなわち，提案手法を用いることにより，単一の抽象度のみを用いる場合に比べ，モデル記述コストを約 45%低減した．

4.3.3　Case Study 2: 患者状態監視

患者状態監視アプリケーションでは心拍数と血中酸素濃度を計測・収集する．この WSN アプリケーションはアメリカ合衆国ミズーリ州最大の病院である Barnes-Jewish 病院に配備され運用された．集められたセンサデータは患者の健康状態確認とリアルタイムでの患者の異常状態を検知するために医師が参照する．

対象とする WSN はデータの計測・送信を担う患者に配備するノード，データ中継用に病院内に配備するノード，データを集計するベースステーションノードで構成される．対象とした患者数は 46 人であり，18 個の中継ノードと 1 つのベースステーションノードを配備し，実運用時間は 41 日を超えている．このアプリケーションを実現するソフトウェアの実装は TinyOS 2.0 上でノードプログラミングを用いて行われている．WSN を設置し，WSN ソフトウェアを実行する環境は病院内であり，屋外に比べ容易に電源が確保できるため，データ中継を担うノード，ベースステーションノードには電源が供給されているものとしている．すなわち，電力資源に対する制約は患者が持つノードに対してのみ存在するものとしている．

このソフトウェでは品質改善のため通信設計の変更を行っている．初期配備ではルーティングプロトコルとして Collection Tree Protocol(CTP)[36] を採用している．CTP とは，TinyOS によりサポートされている，ツリー型のトポロジによりデータ収集を行うプロトコルである．しかし，実際の実行結果から，特に患者のノードと中継ノード間での通信について，CTP では通信の信頼性が低く，データ欠損率が高くなることが明らかになった．そこで Chipara らは，患者ノードと中継ノードとの間の通信に，Dynamic Relay Association Protocol(DRAP) という専用の通信プロトコルを設計，実装し用いている．

患者状態監視に対応する開発は 4.2 節にて既に示した．プロトタイプ開発工程におけるデータ計測と収集に関わる設計は図 4.2 で示したよう DataflowML にて実現可能である．品質改善工程におけるルーティングプロトコルの設計変更は，本開発プロセスでは GroupML にて実現可能である．これは，事前に DRAP に対応するテンプレート実装を用

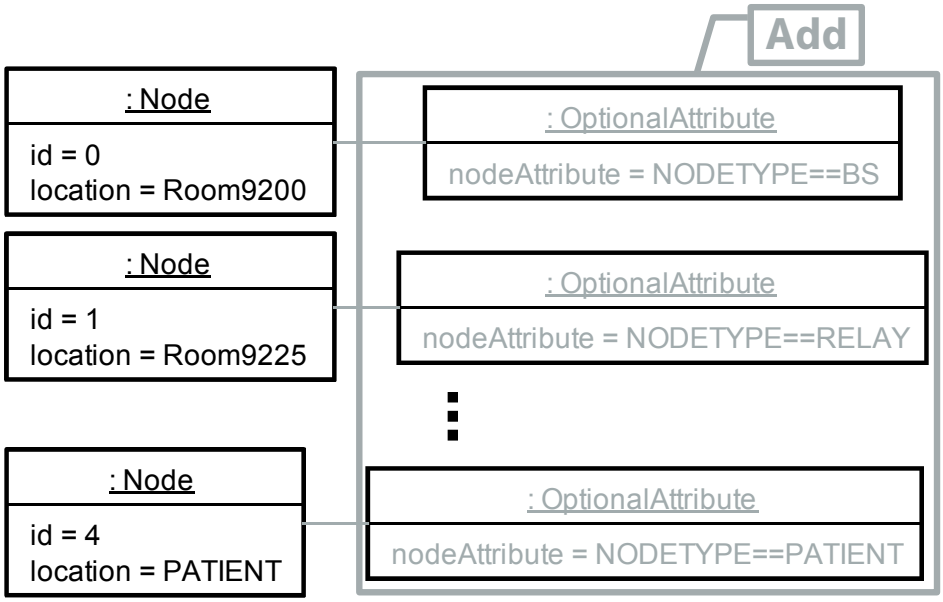


図 4.8: 患者監視ソフトウェアのタスク割当変更のための NetInfo モデルの修正

意した上で，図 4.3 に示したよう `topology` の値を `DRAP` へと変更することで達成可能であった．

タスク割当による品質改善については，WSN を構成するノードに患者用 (`PATIENT`)，中継用 (`RELAY`)，集計用 (`BS`) という種類 (`NODETYPE`) を与え，ノードの種類に応じたタスク割当を実施していた．これは本開発プロセスでは GroupML にて，役割に基づきタスク割当をする様にリーダ・メンバの選択条件を修正することで記述可能であった (e.g. `selectionPattern="NODETYPE==PATIENT"`)．ただし，各ノードがどの `NODETYPE` に該当するかを明示する必要があるため，NetInfo モデルにこの情報を付与した．この変更の様子は図 4.8 に示す．

モデリングステップ数

歴史的建造物監視の場合と同様に，本開発プロセスにおけるモデリングステップ数は，表 4.5 に示すよう，P_{uni} の場合と比べ少ないという結果を得た．まずデータフロー設計については，P_{multi} では DataflowML による記述には 15 ステップ，ノード情報の宣言には 124 ステップをそれぞれ要した．一方 P_{uni} では，HBM ソフトウェアの場合と同様，個々の `Task` と `Role`，`Node` とタスク割当を記述する必要があり 353 ステップを要した．

通信設計ではルーティングプロトコル変更を行っており，P_{multi} では GroupML にて

Group.topology の修正に 2 ステップを要した．一方 P_{uni} では，この修正は Sending/ReceivingTask の 2 箇所の protocol に分散し 4 ステップを要した．

役割に応じたタスク割当は，P_{multi} では GroupML による修正と NetInfo モデルへの情報の付与により達成しており，計 45 ステップでこのタスク割当を達成した．一方 P_{uni} では，ノードと役割との関連を個別に修正する必要があり 58 ステップを要した．

以上をまとめると，$ModelingStep_{uni}$ が 415 であるのに比べ，$ModelingStep_{multi}$ は 186 という結果を得た．すなわち，提案手法の適用によりモデル記述コストを約 55%低減するという結果を得た．

4.3.4　その他のケーススタディ

前述の 2 つのケーススタディと同様，残り 3 つのアプリケーションに対してもプロトタイプ開発工程と品質改善工程を実施した．

火災監視は，消防士の活動支援のため，定期的に温度，湿度，気圧のデータを計測しベースステーションへ送る．このアプリケーションは TinyOS 上で NesC を用いて実装されており，品質改善のための特定の設計は採用していない．また，気候監視では，火災の延焼予測などのため温度，湿度に関する情報を計測，送信する．火災監視と同様，気候監視でも特定の設計は採用されていない．すなわち，これら 2 つのアプリケーションを実現するソフトウェアは DataflowML のみでモデル化可能である．

橋梁監視アプリケーションでは，64 個のノードを用いゴールデンゲートブリッジの健康状態を監視する．センサデータとして加速度を扱い，各ノードにて 20 回の計測ごとに平均値を取り代表値として送信する．当該 WSN は大規模なマルチホップネットワークとなるため，新たに Straw というルーティングプロトコルを提案している．

これら 3 つのケースについて，本開発プロセスはデータ処理設計及び通信設計をモデル化可能であった．プロトタイプ開発工程では，P_{multi} ではデータフローのみに焦点を当てモデルを記述するのに対し，P_{uni} では個々のノード，タスク割当まで考慮してモデルを記述することとなった．よって，全てのケースにおいてプロトタイプ開発に必要となるモデリングステップは P_{uni} に比べ P_{multi} が少ないという結果であった．品質改善工程では，橋梁監視における通信設計の変更のみを扱い，患者監視の場合と同様，P_{multi} では GroupML にて 2 ステップのモデル修正を，P_{uni} では 4 ステップのモデル修正をそれぞれ要した．

表 4.5 に示すように，全体でのモデリングステップ数は，$ModelingStep_{uni}$ に比べ，$ModelingStep_{multi}$ の方が 30-69%低減されていることを確認した．

表 4.6: P_{multi} と P_{expert} における総合モデリングステップ

アプリ名	$\boldsymbol{P_{multi}}$	$\boldsymbol{P_{expert}}$
	$\boldsymbol{MS_{multi}}$	$\boldsymbol{MS_{expert}}$
歴史的建造物監視	178	280
患者監視	186	351
橋梁監視	210	299

熟練者によるモデル化との比較

上記では P_{uni} と P_{multi} がプロトタイプ開発工程後に品質改善工程を繰り返すプロセスを前提とした比較を実施した．しかしながら，開発者が WSN ソフトウェア開発の熟練者である場合，品質改善までを見越してプロトタイプ開発を行うことが可能であると考えられる．したがって本節では，プロトタイプ開発と同時に品質改善のための設計を実施することでモデルを記述するプロセスを P_{expert} とし，これに対してもケーススタディも実施した．すなわち P_{expert} では，NodeML を用い表 4.1 に示したデータ処理設計とともに通信設計，タスク割当設計も同時にモデルへと反映するモデル化を実施する．この際にかかるモデリングステップ $ModelingStep_{expert}(MS_{expert})$ を計測した結果と P_{multi} での結果との比較を表 4.6 に示す．ただし，火災監視，気候監視のケースは特定の通信，タスク割当の設計変更を行っていないため除外している．

この結果から，3 つのケースにおいて，MS_{multi} の方が MS_{expert} よりも少なく，平均して約 38%のモデリングステップを低減している．これは抽象度の使い分けによる関心事の分離と，自動変換による設計変更の伝播によると考えられる．すなわち，本開発プロセスを用いて段階的に設計を決定し，適切な抽象度にてモデルを記述していくことで，熟練者が全ての設計をまとめて記述する場合と比べても少ない記述コストでのモデル化を達成した．

4.3.5 ケーススタディのまとめ

本節では以上のケーススタディの結果をまとめ，研究課題への回答を記す．ここでは研究課題との対応についてのみまとめ，得られた結果に基づく本手法の限界や適用範囲については第 6 に詳述する．

RQ.1 単一の抽象度を用いた場合と比較し，プロトタイプ開発を少ないモデル記述コストで実現できるか

本ケーススタディでは，5 つの WSN ソフトウェアのデータフローを，DataflowML を用いることで表現可能であることを確認した．また表 4.2 はプロトタイプ開発工程にかかるモデリングステップを示している．この結果から，本開発プロセスを用いることで，単一の抽象度を用いる場合よりも，プロトタイプ開発にかかるモデリングステップを平均で約 55%低減した．したがって，本手法は，単一の抽象度を用いた場合と比較し，少ないモデル記述コストでのプロトタイプ開発を実現し，要件 “新規開発における低コストでのプロトタイプ開発が可能であること” を達成する．

RQ.2 現実的な WSN ソフトウェア品質改善のための設計変更を，本開発プロセスにて記述できるか

本ケーススタディでは，5 つの WSN ソフトウェアにて採用されていた通信設計とタスク割当設計を，GroupML，NodeML を使い分けることで表現可能であることを確認した．通信ではルーティングプロトコル及びデータ圧縮アルゴリズムの変更を，実装テンプレートに紐づく語彙を切り替える事でモデル上での表現が可能であった．またタスク割当は，タスクとノード間の関連を任意に変更する事で表現可能であった．したがって，本手法は，現実的な WSN ソフトウェアで実施されていた品質改善を表現可能であり，要件 “通信設計，タスク割当設計を含む細やかな設計変更が可能であること” を達成する．

RQ.3 単一抽象度を用いた場合と比較し，品質改善のための設計変更をより簡潔に記述できるか

RQ2 への回答で示した設計変更にかかるモデリングステップは表 4.3， 4.4 にて示した．この結果から，本開発プロセスを用いることで，単一の抽象度を用いる場合よりも，品質改善のための設計変更にかかるモデリングステップを平均で約 22%低減した．したがって，本手法は，単一の抽象度を用いた場合と比較し，少ないモデル記述コストでの設計変更を実現し，は要件 “品質改善のための設計変更を端的に記述可能であること” を達成した．

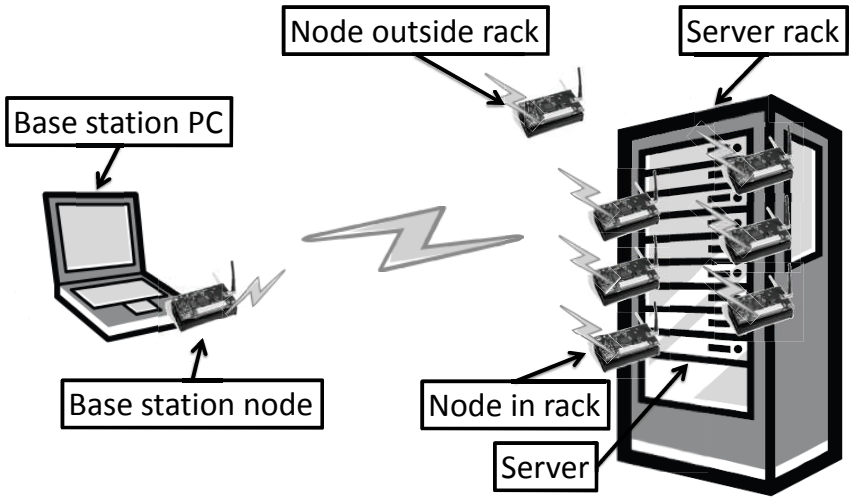


図 4.9: サーバ監視アプリケーションの概要

4.4 ユーザ試験評価

本節では，本研究の目的である品質改善の支援を，現実の開発者に対し可能であるかについて評価する．また，既存の開発プロセスが単一の言語を用いるのに対し，本開発プロセスでは 3 つの DSML を併用するため学習コストは増加しうるこのため，現実の開発者が，現実的な学習時間で本開発プロセスを用いた開発，品質改善が可能かについても評価する．以下ではこれらの評価を，被験者実験を実施した結果を記す．

4.4.1 実験設定

被験者実験は 4 名の開発者による，サーバ近辺の温度を監視する WSN アプリケーションを実現するソフトウェア [80] を，開発を通して実施する．対象とする WSN は図 4.9 に示すよう，サーバラックに設置された 6 つのノード，サーバラック外に設置された 1 つのノード，PC に接続された 1 つのベースステーションの計 8 つからなる．機能要求としては 500 ミリ秒ごとに計測された温度データを収集すること，非機能要求としてはデータ到達率を出来る限り高めることを開発者に提示した．4 名の被験者はいずれも Web アプリケーションなどのソフトウェア開発経験を有し，UML によるモデリングを知っているが，WSN ソフトウェア開発の経験は無い．

被験者実験は以下にまとめた 3 つのイテレーションからなり，各被験者に対しこのイテレーションを実施する．第二イテレーション以降では NodeML を対象外としているが，

表 4.7: 被験者実験における GroupML 上利用可能な通信設計

設計対象	対応するモデル要素	利用可能な語彙
ネットワークトポロジ	topology	TREE (default), STAR, FLAT
データ圧縮	compress	NONE (default), SLZW
データ暗号化	encrypt	NONE (default), AES

これは本実験における対象 WSN ソフトウェアの主たる非機能要求は通信設計により改善可能であるためである．まあ，データ到達率の測定にあたっては NesC コードを生成し，TinyOS を搭載した 8 つの Iris mote [1] を用い，1 時間の実行から測定した．

被験者実験手順:

- 第一イテレーション (プロトタイプ開発工程)

  (1.1) 30 分程度の開発プロセス，DataflowML の詳細，対象 WSN ソフトウェアへの要求の説明

  (1.2) DataflowML を用いたモデル記述

  (1.3) Group-level，Node-level モデル，コードの自動生成と対象環境での実行，データ到達率測定

- 第二イテレーション (品質改善工程)

  (2.1) 15 分程度の GroupML の詳細と設計上の選択肢 (表 4.7) の説明

  (2.2) 第一イテレーションにおけるデータ到達率の提示

  (2.3) 第一イテレーションで自動生成された Group-level モデル対する，GroupML を用いたモデル修正

  (2.4) 対応するコード生成と対象環境での実行，データ到達率測定

- 第三イテレーション (品質改善工程)

  (3.1) 第二イテレーションにおけるデータ損失率の提示

  (3.2) GroupML を用いたモデル修正

  (3.3) 対応するコード生成と対象環境での実行，データ到達率測定

[1] http://www.xbow.jp/zigbee-smartdust.html

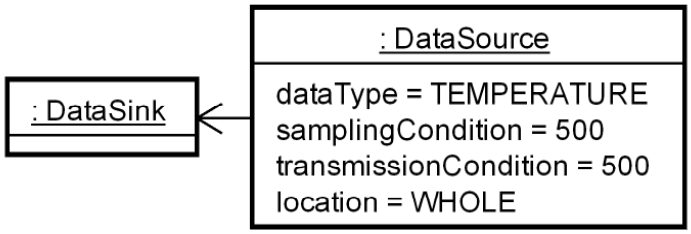


図 4.10: 被験者により記述された Dataflow-level モデル

表 4.8: 各イテレーションにおける被験者ごとのモデリング時間

	モデリング時間 [分]			
被験者	**1st**	**2nd**	**3rd**	合計
被験者 A	3	2	3	8
被験者 B	9	3	12	24
被験者 C	12	16	7	35
被験者 D	7	9	14	30

4.4.2 実験結果

第一イテレーションでは，全ての被験者は同一の Dataflow-level モデルを記述した．このモデルを図 4.10 に示す．また，各被験者がこのモデルを記述するのにかかった時間を表 4.8 の列 “1st” に示す．

第二イテレーションでは，各被験者は品質改善のために異なる設計を選択した．第二，三イテレーションで各被験者が実施した設計変更は表 4.9 及び表 4.10 にまとめた．また，各イテレーションに対応するデータ到達率は図 4.11 に示している．

被験者 A は，第二イテレーションにて `topology` を `STAR` から `TREE` へ，`compression` を `NONE` から `SLZW` へ変更した．またこの設計の再利用に加え，データ送信間隔を表す `transmission condition` を 500 ミリ秒から 5000 ミリ秒へと変更した．この変更は，計測は 500 ミリ秒間隔で行うままであるが，グループリーダへのデータ送信は 5000 ミリ秒間隔で行うことを意味する．被験者 A はこれらの修正を約 2 分で達成した．また，これらの設計変更によりデータ到達率を 94.49%から 99.43%へと向上させた．この結果は，圧縮と送信間隔の伸長により，データの適時性を犠牲に WSN 内通信量を低減させたためであると考えられる．

第三イテレーションでは，グループの構成を階層的なグループから階層関係の無いフ

表 4.9: 被験者 A 及び被験者 B の設計変更

	設計変更	被験者 **A**	被験者 **B**
2nd	グループ構造	変更なし	Hierarchical → flat
	ネットワークトポロジ	STAR → TREE	STAR → FLAT
	データ圧縮	NONE → SLZW	NONE → SLZW
	データ送信間隔	500 → 5000	500 → 10000
3rd	グループ構造	Hierarchical → flat	変更なし
	ネットワークトポロジ	変更なし	FLAT → TREE
	データ圧縮	変更なし	SLZW → NONE
	データ送信間隔	変更なし	10000 → 500

表 4.10: 被験者 C 及び被験者 D の設計変更

	設計変更	被験者 **C**	被験者 **D**
2nd	グループ構造	グループ分割	Hierarchical → flat,
	ネットワークトポロジ	STAR → FLAT	STAR → FLAT,
	データ圧縮	NONE → SLZW	NONE → SLZW,
	データ送信間隔	500 → 5000	500 → 2500
3rd	グループ構造	2 つのグループの統合	変更なし
	ネットワークトポロジ	FLAT → TREE	FLAT → TREE
	データ圧縮	変更なし	変更なし
	データ送信間隔	変更なし	変更なし

ラットなグループへと変更することでデータ到達率を 99.43%から 99.99%へと改善させた．この変更はモデル上のクラスやリンクへの修正を要するが，被験者 A はこれを約 3 分で実現した．

被験者 B は，第二イテレーションにて `topology` を `STAR` か `FLAT` へ，`compress` を `None` から `SLZW` へ変更した．これに加え `transmission condition` を 500 ミリ秒から 10000 ミリ秒に，グループの構成を階層的なグループ構造からフラットなグループ構造へと変更した．被験者 B はこれらの変更を約 3 分で実施した．被験者 C と被験者 D は，`FLAT` トポロジと `SLZW` 圧縮アルゴリズムを用いるという，被験者 B と類似する設計変更を実施した．被験者 C はこれに加えてグループ分割を，被験者 D はグループ構造を階層型からフラット型へと変更した．これらの変更に対し，被験者 C は約 16 分，被験者 D，は約 9 分のモデル編集時間を要した．また，これらの変更により，被験者 B は 96.49%から 62.75%に，

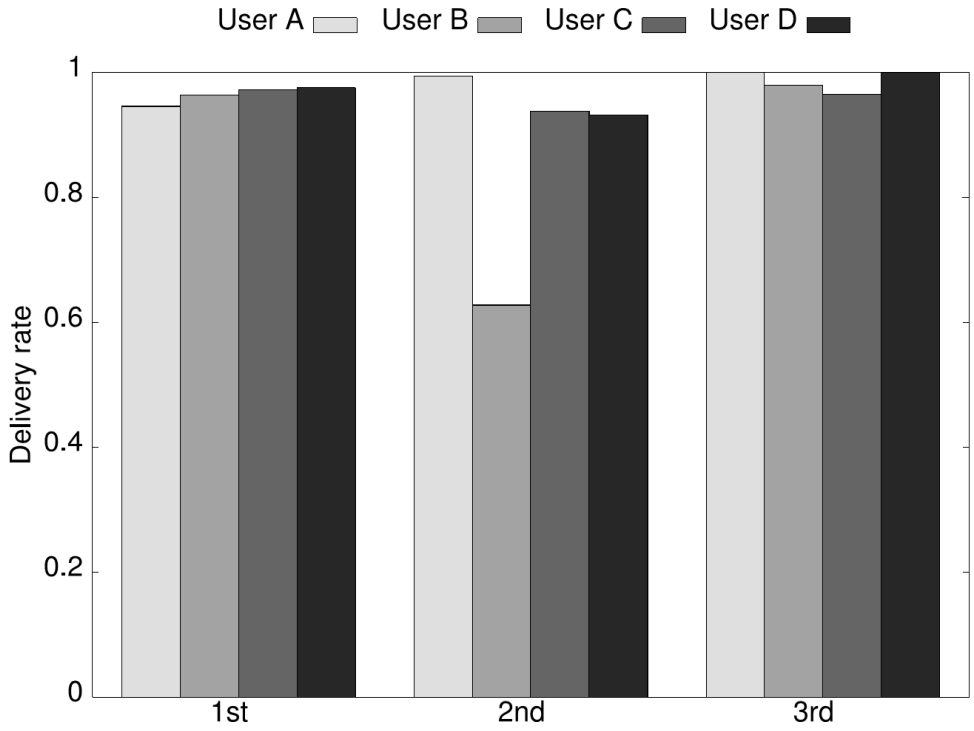


図 4.11: 被験者ごとの各イテレーションにおけるデータ到達率

被験者 C は 97.26%から 93.74%に，被験者 D は 97.56%から 93.17%へとそれぞれデータ到達率を悪化させてしまった．更には，被験者 B と被験者 D の場合においては，実行中に幾つかのノードが停止してしまうという，データ到達率の低下よりも深刻な問題を確認した．データ到達率の悪化やノードの停止は，`FLAT` トポロジに対応する実装であるルーティングプロトコル `Tymo` が，今回の様にノードが密な WSN ではネットワーク中のデータ過多によりうまく機能しなかったためであると考えられる．

しかしながら，第三イテレーションを実施した結果，被験者 B，C，D はデータ到達率をそれぞれ 98.0%，96.59%，100.0%へと改善させ，ノードが停止してしまうケースを排除した．3 名の被験者は主にトポロジを `TREE` へと変更していた．第三イテレーションにおける変更は，被験者 B は約 12 分，被験者 C は約 7 分，被験者 D は約 14 分で実施した．

4.4.3 ユーザ実験のまとめ

以上の結果から，本開発プロセスは，実際の開発者が，WSN ソフトウェアの開発とその品質改善を達成するために適用可能であるといえる．また，GroupML における通信設

計の変更が，WSN ソフトウェアの品質改善を試みる際に有用であることも示した．今回の WSN では FLAT トポロジが適切でなく，実行前に適切な設計を選択することは困難であるが，試行錯誤を繰り返す事で品質改善を達成できることも確認した．被験者実験では，いずれの開発者も，計 30 分程度のプロセスや DSML に対する説明で，各イテレーションを 16 分以内で達成していた．すなわち，本手法を用いることで，現実の開発者が，数十分の学習時間で開発対象に合わせた WSN ソフトウェアを開発可能であった．

4.5　まとめ

本章では，WSN ソフトウェアの新規開発における品質改善を支援するため，低コストでのプロトタイプ開発と細やかなで端的な品質改善を両立する開発プロセスを，多段階 MDD フレームワークを用いることで提案，実現した．提案した開発プロセスでは，工程ごとに抽象度を使い分けることでこれらの要件を達成している．

5 件のケーススタディを通し，実世界で運用実績のある現実的なアプリケーションに対し，これを実現する WSN ソフトウェアのデータフロー，通信，タスク割当設計を，本開発プロセスを用いてモデル化可能であることを示した．また，ケーススタディでは，本開発プロセスにおけるモデリングステップ数が，Node-level MDD に比べ，全体で 30-69%低減されていることを確認した．更にユーザ試験評価により，現実の開発者が，本開発プロセスを通し実際の WSN ソフトウェアのモデル化，品質改善が可能であることも示した．すなわち，本開発プロセスは，開発対象ごとに WSN ソフトウェアの品質を改善する様なソフトウェア開発を支援可能である．

第5章　WSNソフトウェア移植開発のための多段階モデル駆動開発手法

5.1　はじめに

本章ではWSNソフトウェアの移植開発支援に向けた，多段階MDDフレームワークを用いた開発手法を提案する．2.5節で述べた様に，既存のWSNソフトウェアの移植開発のためのMDA手法では，開発対象ごとの品質改善のための要件を満たすことができない．移植開発においては以下の要件が求められる．

- 初期実装獲得工程における要件

  要件2. 移植開発における多様なWSNプラットフォームからの低コストでの移植が可能であること

- 品質改善工程における要件

  要件3. 通信設計，タスク割当設計を含む細やかな設計変更が可能であること

  要件4. 品質改善のための設計変更を端的に記述可能であること

既存のMDA手法では単一の抽象度のPIMを用いた移植開発支援を提案している．抽象度の高いPIMを用いるMDAでは，抽象化のためモデルに含まれる設計情報が限定的であり要件3を達成できない．一方，抽象度の低いPIMを用いるMDAでは，関心事の混在によりモデル上での記述が複雑となり要件4を達成できない．

更に，複数の抽象度に分類可能な，多様なプラットフォームを持つWSNドメインでは，移植元プラットフォームとPIMとの抽象度が異なる場合，移植元PSMが持つ設計情報と，PIMが持つ設計情報との間に乖離が生じる．この結果，抽象度の低いPIMでは，移植元のPSMの抽象度が高い場合，PIMが持つべき通信やタスク割当に関する設計情報が移植元PSMに含まれない．このような設計情報は開発者による意思決定に基づき人手で補完されるためコストの増加につながり要件2を達成できない．すなわち，単一の抽象度

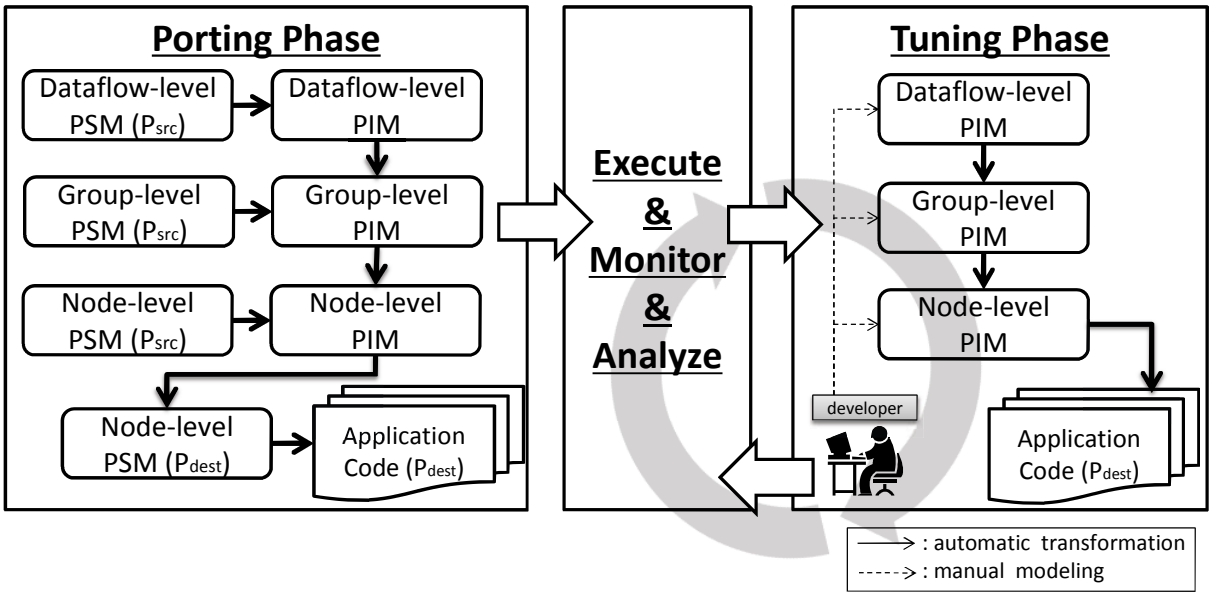


図 5.1: 移植開発のための開発手法の全体像

のみを扱う既存研究では，抽象度間のトレードオフや PIM，PSM 間の乖離のため，上述の要件を全て同時に満たすことができない．

そこで本研究では，複数の抽象度を併用する MDD フレームワークを移植開発支援への適用について論じる．開発手法の全体像を 5.2 節にて示し，その構成要素と具体例についても示す．続く 5.3 節では提案手法の妥当性確認のためのケーススタディについて詳述し，最後に 5.4 節にて本章をまとめる．

5.2　WSN ソフトウェア移植開発のための多段階 MDD 手法

図 5.1 に多段階 MDD フレームワークを用いた，多様なプラットフォームからの移植性向上のためのソフトウェア開発手法の全体像を示す．異なる抽象度を提供する多様なプラットフォームを対象とするため，本開発手法は DataflowML，GroupML，NodeML で記述されたモデルをそれぞれ Dataflow-level PIM，Group-level PIM，Node-level PIM と呼び，3 種の PIM を併用する．

特定のプラットフォームからの移植開発の際には，まず移植工程にて，開発者は移植元となるプラットフォームを決定し，移植元となるプラットフォームが属する抽象度の PIM を同定する．その後，移植元プラットフォームに対応する PSM から PIM への変換 (以下，PSM2PIM 変換) によりプラットフォームに非依存な情報を抽出，PIM として生成

する．プラットフォームに対応する抽象度の同定には表 2.3 で記した分類を用いる．例えば，TikiriDB は Dataflow-level に属するプラットフォームであるため，TikiriDB の PSM は Dataflow-level PIM へと変換することで移植に利用可能となる．この際，対象とするプラットフォームに対応する PSM2PIM 変換が存在しない場合はこれを開発する必要が生じる．この変換は同抽象度のモデル間の変換であり，プラットフォーム依存な要素を含むモデルをプラットフォーム依存な情報を含まないモデルへと対応付ける抽象化変換である．

生成された PIM はフレームワークを通して下位の抽象度の PIM，移植先プラットフォームに対応する PSM，移植先プラットフォームで実行可能なコードへと順次変換，詳細化される．プラットフォームの抽象度が Node-level よりも上位である場合，移植元 PSM に含まれないが移植先 PSM には含まれる設計情報が存在しうるが，これらはフレームワーク内のモデル変換により自動で補完され，人手での意思決定やモデル修正は不要である．これにより，“要件 2. 多様な WSN プラットフォームからの低コストでの移植” を達成する．

続く品質改善工程では，移植先プラットフォームに対する適応のための品質改善を実施する．この際本開発手法では，PIM は移植工程にて自動生成されたモデルを再利用することで適応を達成する．開発者は任意の PIM 上で設計を変更し，モデル変換とコード生成を通し下位へのその設計を伝播させることが可能である．これにより新規開発における品質改善工程と同様，抽象度の DSML の併用によって “要件 3. 細やかな設計変更” と “要件 4. 端的な設計変更の記述” の両立を達成する．

本研究では具体例として，Dataflow-level に属するプラットフォームである TikiriDB と，Node-level に属するプラットフォームである TinyOS/NesC を対象としている．これらの設計，実装については 3.3 節に記した．

5.3 ケーススタディ

本開発手法の適用可能性を確認するため，サーバ監視ソフトウェア [80] を対象としたケーススタディを実施した．これは 4.4 節で用いたソフトウェアと同様であり，毎秒温度データを収集するものである．既存の WSN ソフトウェアとして，TikiriDB を対象プラットフォームとして開発された成果物が存在する前提のもと，TinyOS/NesC という異なるプラットフォームに向けて同一の機能性を実現する WSN ソフトウェア開発を行うことを考える．本ケーススタディに対応する研究課題を以下に示す．

RQ.1 単一の抽象度を用いた場合と比較し，少ないコストでの移植を実現できるか

RQ.2 本開発手法を用いた設計変更が，現実世界での品質改善を実現できるか

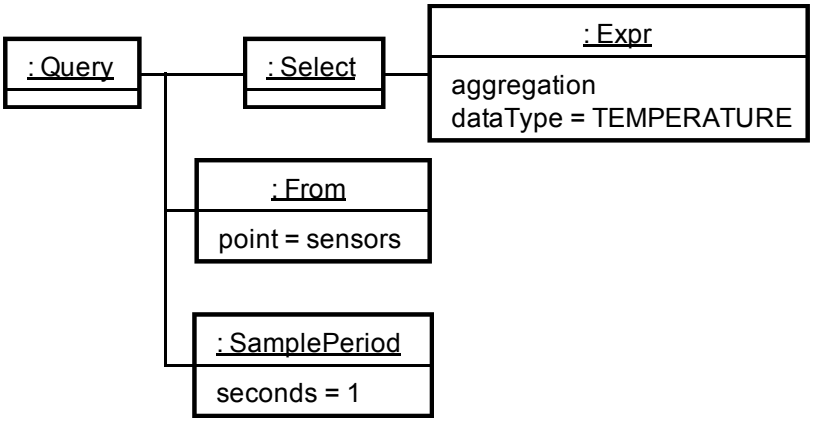


図 5.2: TikiriDB PSM によるサーバ監視ソフトウェア

RQ1 は要件 “移植開発における多様な WSN プラットフォームからの低コストでの移植が可能であること” に対応する．品質改善工程は第 4 章の場合と同様であり，**??**節で示した様に要件 “通信設計，タスク割当設計を含む細やかな設計変更が可能であること” と要件 “品質改善のための設計変更を端的に記述可能であること” を達成する．そこで，RQ2 では，本手法を用いた移植開発が，移植先プラットフォームへの適応のための品質改善に寄与するか否かについて，実機を用いた実験により調査する．なお，WSN ソフトウェアに品質測定にあたり，本ケーススタディではデータ到達率とエネルギー消費量という 2 つの重要な品質に関わる要素 [16] を，半日間の実行を通して計測した．

5.3.1　移植工程

本節では TikiriDB にて開発された既存のサーバ監視ソフトウェアを TinyOS/NesC へ移植するシナリオを，提案手法を用いて実施した結果を記す．更に，移植前後の 2 つのプラットフォームで WSN ソフトウェアを実行し、品質を測定した結果も示す．

移植の実施結果

図 5.2 に，移植元となる，TikiriDB 上で実現されたサーバ監視ソフトウェアのモデルを記す．このモデルでは，対象ソフトウェアは 1000 ミリ秒間隔で温度データを計測，送信することが示されている．

移植工程では，まずフレームワークは TikiriDB PSM を入力とし，3.3 節で述べた PSM2PIM

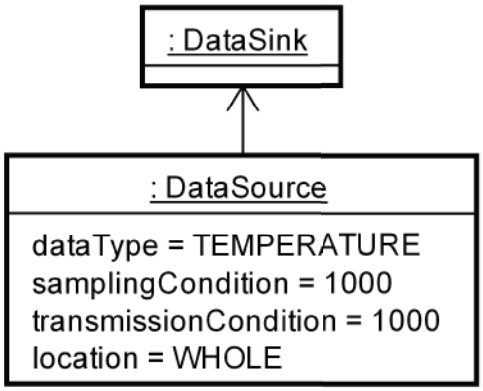


図 5.3: Dataflow-level PIM によるサーバ監視ソフトウェア

変換により Dataflow-level PIM を生成する．これは，TikiriDB が Dataflow-level に分類されるプラットフォームであるためである．生成された Dataflow-level PIM を図 5.3 に示す．

続いて取り込まれた既存ソフトウェアを TinyOS/NesC へ移植することを考える．フレームワークは自動変換により，Group-level PIM と Node-level PIM を Dataflow-level PIM から生成する．更に，Node-level PIM は TinyOS/NesC に対応する PSM へと変換され，TinyOS/NesC PSM は NesC コードへと変換される．この際，TikiriDB に含まれない通信やタスク割当の設計情報は PIM2PIM 変換により自動で補完されるため，人手でのモデル修正は不要である．これにより，TikiriDB から TinyOS/NesC への移植を達成する．

移植コストの比較

提案手法が，単一の PIM のみを用いた場合と比べ低コストでの移植が可能なことを確認するため，同一アプリケーションに対し，3 つの移植手法を用いたケーススタディを実施した．ただし，本ケーススタディでは単一の PIM として Node-level PIM を用いるものとする．これは，Dataflow-level PIM または Group-level PIM のみを用いる場合細やかな品質調整を達成できないためである提案手法の比較対象として，Node-level PIM に不足する情報を手動で補完する手法と自動で補完する手法が考えられる．よって，まず TikiriDB PSM に含まれるデータ処理に関する設計情報のみを Node-level PIM へと対応付け，不足する情報は手動で決定，補完する開発手法 (P_{uni_man}) を用いた．また，TikiriDB PSM を Node-level PIM へと対応付ける際に，不足する設計情報を PSM2PIM 変換にて自動補完する開発手法 (P_{uni_auto}) も用いた．これら 2 つの手法と，提案手法である多段階 PIM を用いる開発手法 (P_{multi}) とで，移植にかかるコストを比較する．3 つの手法の概要を図 5.4

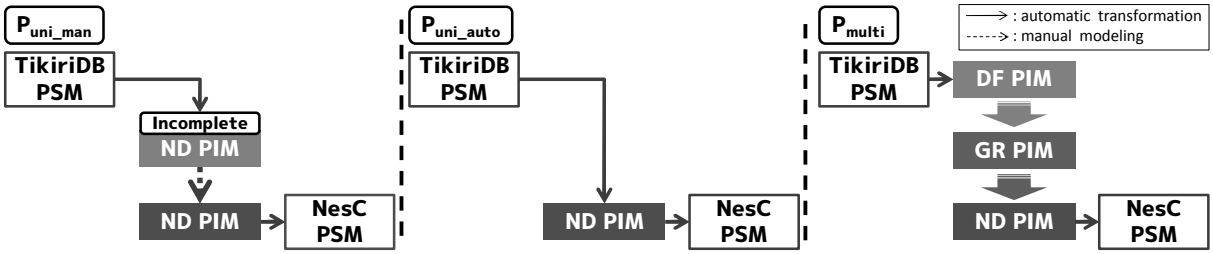


図 5.4: 移植コスト比較に用いる開発手法の概要

に示す.

比較に辺り，各手法は下記の 3 つの変換を事前に開発する必要がある.

変換 1. TikiriDB PSM から PIM への変換

変換 2. Node-level PIM から TinyOS/NesC PSM への変換

変換 3. TinyOS/NesC PSM から NesC コードへの変換

移植コストの比較として，変換 1 の開発にかかるコストを比較する．これは，変換 2，3 はいずれの手法においても同一のものを用いるためである.

変換 1 については，P_{uni_man} では TikiriDB PSM に含まれる情報のみを用い，部分的な Node-level PIM を生成する変換と，生成された PIM に対する手動での修正からなる．TikiriDB PSM から Node-level PIM への変換では，TikiriDB 上に Node-level PIM が含むべきタスクアーキテクチャ設計 (タスク分割設計とタスク間通信設計) と詳細設計 (タスク割当設計) が含まれていない．このため，P_{uni_man} の変換ではデータの計測，集約，送受信に対応するタスクのみを生成する．生成された不完全な PIM に対し，開発者は移植の度にタスク分割設計，タスク間通信設計，タスク割当設計をそれぞれ実施，モデルに反映する．これにより図 5.4 左部に示す開発を達成する．P_{uni_auto} では，変換 1 は TikiriDB PSM から，不足する設計情報を補完しつつ完全な Node-level PIM を生成するものとなる．この変換では TikiriDB 上にないタスク分割設計，タスク間通信設計とタスク割当設計の情報を付与している．P_{multi} では，変換 1 は TikiriDB PSM から Dataflow-level PIM への変換である．これらの変換は事前に一度開発され，移植の度に必要な作業はない．これにより図 5.4 中央部，右部に示す開発をそれぞれ達成する．以上をまとめると，移植コストは，ATL 変換の開発コストを C_{trans}，手動でのモデル修正コストを C_{mod} とすると以下のように表される.

$$Cost\ of\ porting = C_{trans} + C_{mod}$$

表 5.1: 各種法における PSM2PIM 変換実装の LOC と XML 表現上でのサイズ

手法	**LOC**	**XML size**
P_{uni_man}	74	614
P_{uni_auto}	194	1772
P_{multi}	86	764

C_{trans} の指標としては，まず ATL 実装の空行，コメントを除くコード行数 (LOC: Lines of Code) を用いる．LOC はソフトウェア成果物のサイズを表し，開発コストの推定する指標として用いられるためである [24]．C_{mod} は自動生成された不完全なモデルへの修正コストであり，これは 4.3 と同様にモデリングステップ (MS) を用いる．

また，C_{trans} と C_{mod} を等価に扱うための異なる C_{trans} の指標として，ATL 変換と NodeML の XML 表現におけるサイズを用いる．XML 上でのサイズとは，XML 文書中のノード (Element と Attribute) の数とした．ATL 変換の XML 表現は，旧バージョンの ATL に組み込まれているモデル抽出器にて獲得可能であり，LOC とともにソフトウェアのサイズを表す．NodeML で記述されたモデルは EMF 上にて XML 形式で表現されるよう実装されている．C_{mod} は，生成された不完全なモデルを，P_{uni_auto}，P_{multi} で自動生成されたモデルと同等のモデルにするまでにかかる，XML 表現上でのノード数を表す．

手動でのモデル修正は移植元のソフトウェアごとに異なるため幾つかの例を用い C_{mod} を計測した．データ処理やデータの種類の追加に対する拡張性を調査するため，前節で用いたサーバ監視ソフトウェア ($SeverApp_{temp}$) を拡張した 2 種類のソフトウェアを用いた．すなわち、対象ソフトウェアは，温度データの計測，送信に加え，平均値計算処理も行うソフトウェア ($SeverApp_{avg\_temp}$)，温度データに加えて湿度データも計測，送信するソフトウェア ($SeverApp_{temp\_humid}$) とした．更に，ノード数の増加に対する拡張性を調査するため，対象 WSN に含まれるノード数を 4，8，16，32 個とした場合それぞれについての比較も実施した．

以下では，上述の実験設定に基づき実施された実験結果について記す．表 5.1 は変換 1 の LOC 測定結果を示している．LOC の測定結果より，P_{uni_auto} の LOC は、P_{multi} の LOC に比べ約 2.26 倍となり，PSM2PIM 変換の開発コストは P_{multi} の方が P_{uni_auto} に比べ低くなるという結果を得た．また，P_{uni_man} は，P_{multi} に比べ約 0.86 倍の LOC となり，P_{multi} における開発コストが，P_{uni_man} における開発コストに比べて高くなるという結果を得た。

しかしながら，P_{uni_man} では移植の度に手動でのモデル修正が必要となる．この際にかかる C_{mod} を，ソフトウェアとノード数の組合せごとに計測した結果を表 5.2 に示す．ま

表 5.2: P_{uni_man} における手動修正にかかるモデリングステップと XML 表現上での追加ノード数

ソフトウェア名＼ノード数		**4**	**8**	**16**	**32**
$SeverApp_{temp}$	MS	33	37	45	61
	XML size	57	73	105	169
$SeverApp_{avg\_temp}$	MS	50	54	62	78
	XML size	60	76	108	172
$SeverApp_{temp\_humid}$	MS	57	65	81	193
	XML size	97	129	193	321

た，P_{uni_man} における手動でのモデル修正の結果を，ノード数 4 の場合での $SeverApp_{temp}$ を用いて例示する．図 5.5 では生成された Node-level PIM と，手動で加えられた修正を示している．P_{uni_man} における PSM2PIM 変換では，`SamplingTask`，`SendingTask`，`ReceivingTask` と，各々の既知の属性のみを生成する．これに対し開発者は，タスク分割設計に当たる必要な `Task` の同定と `Role` と関連する `Task` の決定，タスク間通信設計に当たる `protocol` などの属性の決定，タスク割当設計に当たる `Role` と `Node` 間の関連の決定をそれぞれ実施し，モデルを修正する．図 5.5 ではタスク分割設計に 13 ステップ，通信設計に 12 ステップ，タスク割当設計に 8 ステップを要し，合計モデリングステップとして 33 を要した．これは，小規模な WSN にてデータ計測のみを実施する WSN ソフトウェアを新規に開発する際にかかるモデリングステップとほぼ同等である．

上記の様な手動でのモデル修正を 3 種のソフトウェアと，ノード数が変化した場合の全てに対し実施した．表 5.2 に示す結果から，対象とするソフトウェアのデータ処理が複雑になるほど手動でのモデル修正コストが増加することがわかる．また，ノード数の増加によってもモデル修正コストが増加する事を確認した．この様な修正は P_{uni_auto} 及び P_{multi} では不要であるが，P_{uni_man} では移植の都度必要となり移植コストの増加につながる．

以上をまとめた XML 表現における移植コストを表 5.3〜 5.5 に示す．移植コストは，PSM2PIM 変換開発コストの比較からから P_{multi} の方が P_{uni_auto} に比べ低くなる．また P_{uni_man} は，P_{multi} に比べ PSM2PIM 変換開発コストは低いが，移植の都度手動でのモデル修正が必要であり，移植を実施する度にコストが増加する．表 5.3〜 5.5 の結果から，いずれのソフトウェアにおいても，ノード数が 32 の場合，P_{uni_man} の移植コストが P_{multi} の移植コストを越える．また，最も単純なノード数 4 の $SeverApp_{temp}$ の場合においても，移植を 3 度実施することで P_{uni_man} の移植コストが P_{multi} の移植コストを越える．したがって，中規模の WSN の対象とする場合，移植を幾度も実施する場合には，移植コスト

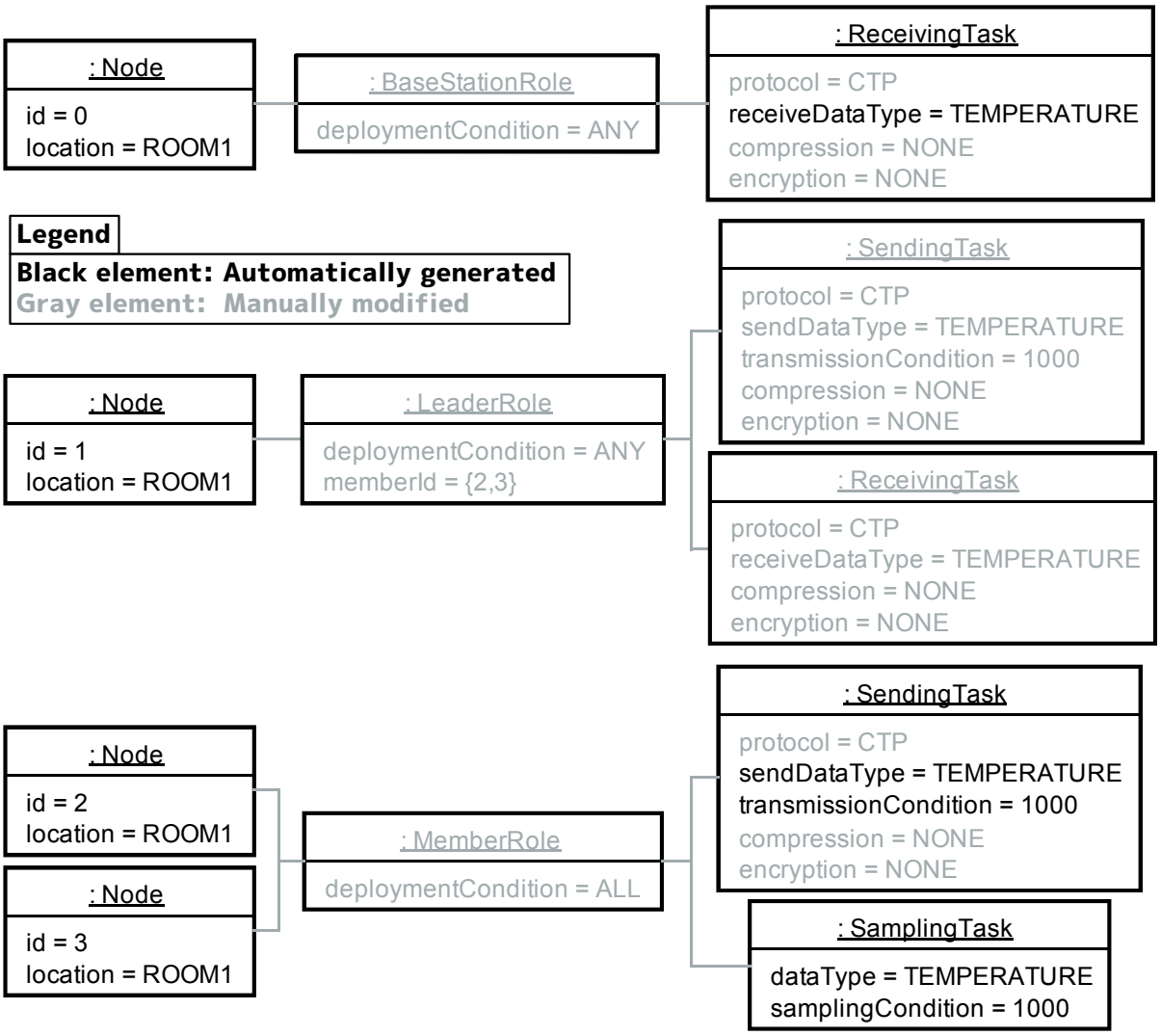


図 5.5: P_{uni_man} における $SeverApp_{temp}$ の自動生成モデルと手動修正

表 5.3: 各手法における移植コスト ($SeverApp_{temp}$)

手法	C_{trans}	C_{mod}	移植コスト
P_{uni_man} (4 nodes)	614	57	671
P_{uni_man} (8 nodes)	614	73	687
P_{uni_man} (16 nodes)	614	105	719
P_{uni_man} (32 nodes)	614	169	783
P_{uni_auto}	1772	0	1772
P_{multi}	764	0	764

表 5.4: 各手法における移植コスト ($SeverApp_{avg\_temp}$)

手法	C_{trans}	C_{mod}	移植コスト
P_{uni_man} (4 nodes)	614	60	674
P_{uni_man} (8 nodes)	614	76	690
P_{uni_man} (16 nodes)	614	108	722
P_{uni_man} (32 nodes)	614	172	786
P_{uni_auto}	1772	0	1772
P_{multi}	764	0	764

は P_{uni_man} に比べ P_{multi} の方が低くなると考えられる．

上記の移植コストは P_{uni_man} における不完全なモデルを，その他のプロセスにて自動生成されるモデルと同等なモデルとするまでにかかるモデル修正コストを用いている．しかしながら，開発者が WSN ソフトウェアの熟練者である場合，自動生成されるモデルよりもより簡潔で同一の要求を達成するモデルが記述可能であることが考えられる．例えば，$SeverApp_{temp}$ において対象とする WSN が小規模である場合には中継ノードが不要となることが考えられる．この場合，熟練した開発者はタスク分割設計において図 5.5 で示した `LeaderRole` とそれに付随する送受信タスクを不要とし，図 5.6 に示す様なモデルへと修正を施す．この場合，修正に要するモデリングステップは 17 となり，自動生成されたモデルへの修正の約半分での修正を達成している．

この様に移植時のモデル修正コストは，手動で設計情報を決定，付与する開発者にも依存することが考えられる．特に開発者が WSN ソフトウェア開発に精通している場合，このコストの低減が見込まれる．しかしながら，P_{uni_man} の場合，設計情報の不足からモデル修正は必須であり，これが移植の度に必要となる．これに対し P_{multi} では多段階抽象度 PIM と自動変換による設計情報の補完により，熟練者が P_{uni_man} を用いた場合であって

表 5.5: 各手法における移植コスト ($SeverApp_{temp\_humid}$)

手法	C_{trans}	C_{mod}	移植コスト
P_{uni_man} (4 nodes)	614	97	711
P_{uni_man} (8 nodes)	614	129	743
P_{uni_man} (16 nodes)	614	193	807
P_{uni_man} (32 nodes)	614	321	935
P_{uni_auto}	1772	0	1772
P_{multi}	764	0	764

表 5.6: データ到達率: TikiriDB vs. TinyOS/NesC

プラットフォーム	送信データ数	受信データ数	データ到達率
TikiriDB	38400	37597	97.906%
TinyOS/NesC	43200	42768	99.000%

も生じる移植のためのモデル修正コストの排除に成功している．

移植前後での品質測定の結果

$SeverApp_{temp}$ について，移植後は WSN ソフトウェアを実機上で実行し，TikiriDB と TinyOS/NesC のそれぞれについてデータ到達率とエネルギー消費量を計測した．これにより，対象とするプラットフォームごとの品質変化と品質改善の必要性を確認する．

表 5.6 に，半日に渡る実行により得られた，計測と送信を担った 3 つのノードにおけるデータ到達率の平均値を示す．TinyOS では 43200 パケットを，TikiriDB では 38400 パケットをそれぞれ送信している．これは，入手可能な TikiriDB [1] の実行コードでは，一定の間隔で計測に遅延が発生するためである．TikiriDB 上では 97.9%のデータ到達率であったのに対し，TinyOS 上では 99%のデータ到達率であった．TikiriDB では通信時にブロードキャストを用いることがミドルウェア側で決定されているのに対し，TinyOS では信頼性の高い CTP を採用したため，TinyOS におけるデータ到達率の方が TikiriDB に比べが高かったと考えられる．

エネルギー消費量については，各ノードにて，実行前後のバッテリ電圧を測定した．バッテリ電圧はエネルギー消費量を直接表すメトリクスではないものの，指標の一つとして有効である．表 5.7 に示す電圧の値は，電圧の計測値が変動しやすいため 30 回の計測値の

[1] https://github.com/scorelab/TikiriDB

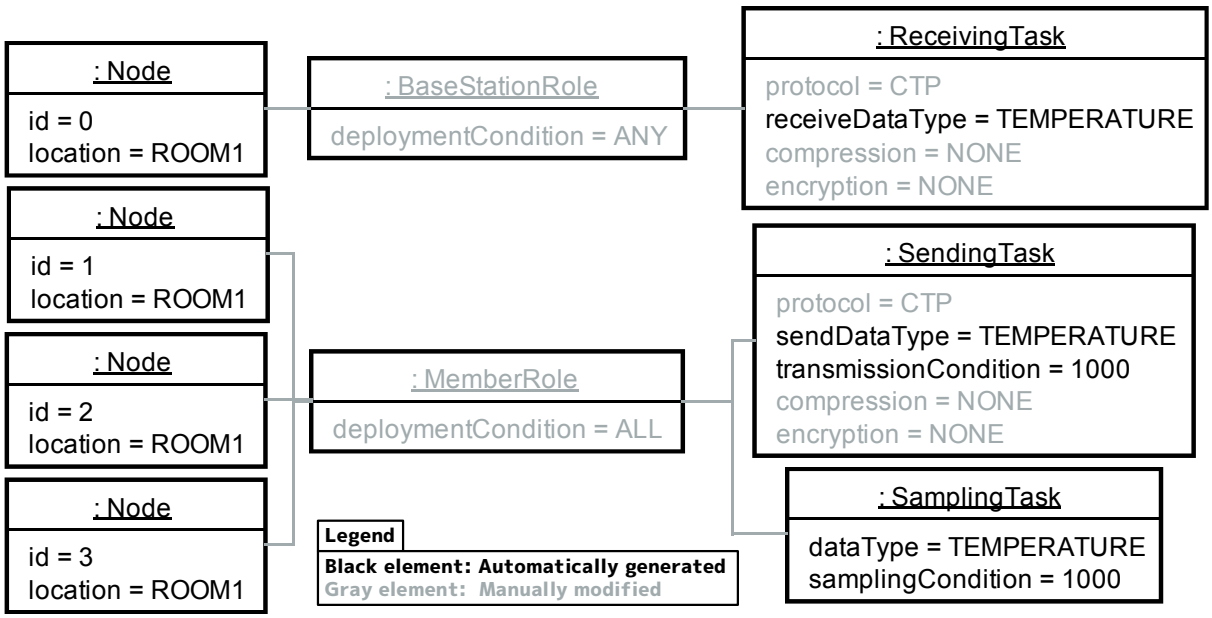


図 5.6: 熟練者による P_{uni_man} における $SeverApp_{temp}$ の自動生成モデルへの手動修正

平均を用いている．この表から，TinyOS/NesC における電圧降下の方が，TikiriDB における電圧降下よりも大きいことがわかる．CTP にはネットワーク内にてトポロジを形成，維持するためのオーバヘッドがあるため，今回の実行においてはブロードキャストを用いる TikiriDB の方が，エネルギー消費量が少なかったと考えられる．

これらの測定結果から，同一の機能性を実現する場合であっても，プラットフォームごとに品質が変化しうることがわかる．TinyOS/NesC における消費電力量の増加は，移植先を考慮せず，単に WSN ソフトウェアを移植することは，品質を保つ上で十分でないことを示唆している．故に本開発手法は，PIM 上での設計変更により，新たに対象とする移植先のプラットフォームに WSN ソフトウェアを適応させる品質改善工程も考慮する．

5.3.2　品質改善工程

WSN ソフトウェアの移植後は，必要に応じて設計変更を実施し，その品質を改善，移植先のプラットフォームへ適応させる．本開発手法では，PIM 上にて，ソフトウェアの高抽象度での設計変更を実施することで品質改善が可能である．よって，本ケーススタディでは，移植時に自動生成された PIM に対して設計変更を実施し，これを反映する WSN ソフトウェアを実行，その性能を測定した．

表 5.7: バッテリ電圧値: TikiriDB vs. TinyOS/NesC

プラットフォーム		初期 電圧 [V]	実行後 電圧 [V]	電圧 降下 [V]
TikiriDB	Node 1	2.719	2.585	0.135
	Node 2	2.666	2.569	0.097
	Node 3	2.721	2.619	0.102
TinyOS/NesC	Node 1	2.941	2.649	0.291
	Node 2	2.914	2.628	0.286
	Node 3	2.885	2.610	0.275

適応の実施結果

本ケーススタディでは，データ到達率の向上と電力消費量の低減を目的に，通信時のデータ圧縮を採用することを考える．データ圧縮を用いることで，通信量の削減により通信コストの低減が，通信頻度の伸長によりパケット衝突によるデータ損失の低減がそれぞれ見込める．

提案手法では，この設計変更は Group-level PIM，Node-level PIM の双方で実現可能である．本ケーススタディでは Group-level PIM を用いて上述の設計変更を実施した．これは Group-level PIM 上で設計変更を行うことで、Node-level では各タスクに分散する通信設計を，グループ内通信として端的に記述可能なためである．具体的には，自動生成されたモデルでは圧縮アルゴリズムを採用していなかったものを，WSN 用圧縮アルゴリズム SLZW [78] に変更した．これは Group-level PIM 上では，各 `Communication` の属性 `compression` の値を `NONE` から `SLZW` に変更することで達成する．この様子を図 5.7 に示す．

移植前後での品質測定の結果

前述のデータ圧縮を用いる設計変更により，実際に WSN ソフトウェアの品質が改善され，移植先プラットフォームに適応することを確認するため，設計変更前後の WSN ソフトウェアを実機上で実行しその品質を測定した．設計変更前後の PIM から生成された実装を用い，データ到達率とエネルギー消費量を，半日の実行を通して 4 つの Mica-z mote [2] を用いて実機上で計測した結果を以下に記す．

まず表 5.8 に，計測と送信を担った 3 つのノードにおけるデータ到達率の平均値を示す．

[2] http://www.xbow.jp/zigbee-smartdust.html

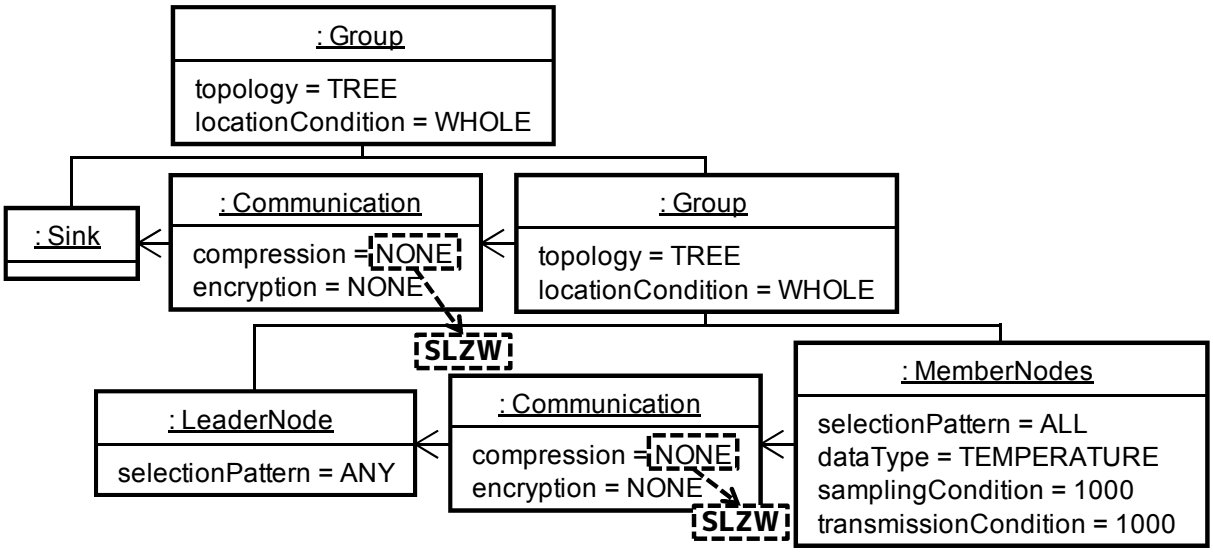


図 5.7: Group-level PIM における圧縮アルゴリズムの修正

表 5.8: データ到達率: No compression vs. SLZW

選択した設計	送信データ数	受信データ数	データ到達率
No compression	43200	42768	99.000%
SLZW	43200	43199	99.998%

この結果から，SLZW アルゴリズムを用いることでデータ到達率の改善が達成されたことがわかる．これは，送信間隔の伸長によりパケットの衝突が低減されたこと，総計での送信データ量が低減されたことによると考えられる．

これに加え，バッテリ電圧に対しては，圧縮を用いない場合の方が，SLZW を用いる場合に比べ電圧の低下が早い事を確認した (図 5.8)．これは，SLZW を用いる場合の方が，通信量が少ないため電力の節約につながったと考えられる．すなわち，SLZW は，圧縮を用いない場合に比べエネルギー消費量が少ないと考えられる．

以上より，本ケーススタディではデータ圧縮を用いることでデータ到達率とエネルギー消費量の双方の改善に成功した．また，本開発手法を通し，WSN ソフトウェアが，移植先の新たな環境に対して適応可能であることを確認した．

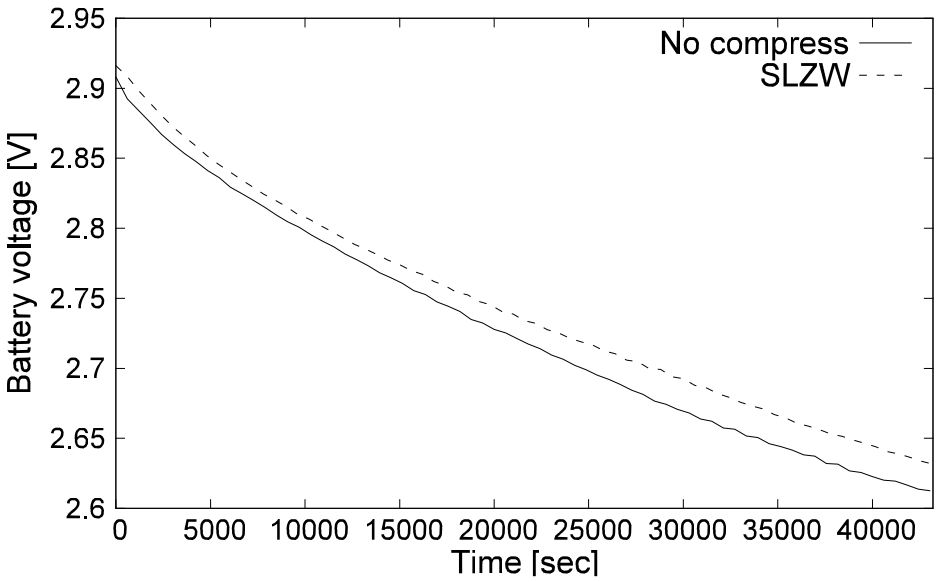


図 5.8: 計測ノードの平均バッテリ電圧の推移

5.3.3 ケーススタディのまとめ

本節では以上のケーススタディの結果をまとめ，研究課題への回答を記す．ここでは研究課題との対応についてのみまとめ，得られた結果に基づく本手法の限界や適用範囲については第 6 に詳述する．

RQ.1 単一の抽象度を用いた場合と比較し，少ないコストでの移植を実現できるか

本ケーススタディでは，サーバ監視ソフトウェアを対象に，既存の TikiriDB PSM を，TinyOS/NesC 上に移植する事で，抽象度の異なるプラットフォーム間の移植を，提案手法を用いて実現した．この際，多段階 PIM を用いる P_{multi} において移植のための PSM2PIM 変換開発コストは，LOC の測定結果より，Node-level PIM のみを用い，不足する設計情報を人手で決定，補完する P_{uni_man} の約 1.16 倍であった．しかしながら，P_{uni_man} では移植の都度必要となる人手でのモデル修正コストがかかる．このモデル修正コストは対象ソフトウェアが複雑化するほど増加し，移植を繰り返す度に増加する．更に，最も単純なソフトウェアを，小規模な WSN を対象に移植する場合においても，新規開発にかかるモデル記述コストと同程度のモデル修正が必要となることを確認した．変換開発コストとモデル修正コストを合計した結果より，最も単純なソフトウェアの場合においても，ノード数 32 個の中規模の WSN を対象とする場合や，移植を 3 回以上実施する場合，P_{uni_man} を

用いる場合に比べ，P_{multi} を用いた方が，低コストでの移植を実現するといえる．

また，P_{multi} における PSM2PIM 変換開発コストは，Node-level PIM のみを用い，不足する設計情報を変換規則により補完する P_{uni_auto} の約 0.44 倍であった．したがって，P_{uni_auto} を用いる場合に比べ，P_{multi} を用いた方が，低コストでの移植を実現するといえる．

以上の結果より，本手法は，単一の抽象度を用いた場合と比較し，少ないコストでの移植を実現し，要件 “移植開発における多様な WSN プラットフォームからの低コストでの移植が可能であること” を達成する．

RQ.2 本開発手法を用いた設計変更が，現実世界での品質改善を実現できるか

本ケーススタディでは，TikiriDB プラットフォームと TinyOS/NesC プラットフォームとでの WSN ソフトウェアの品質測定を行った．これにより，開発対象とするプラットフォームごとに適応のための設計変更が必要となることを示した．

また，TinyOS/NesC プラットフォームを対象に，提案手法を用いて適応のための設計変更を実施し，設計変更前後での品質測定を実施した．この結果，データ到達率，電力消費量の双方の品質を改善した事を確認した．これにより，本手法は，現実世界での品質改善を実現し，開発対象となるプラットフォームごとの品質改善を支援が可能である事を確認した．

5.4　まとめ

本章では，WSN ソフトウェアの移植開発における品質改善を支援するため，低コストでの移植と細やかなで端的な品質改善を両立する開発手法を，多段階 MDD フレームワークを用いることで提案，実現した．本手法では複数の抽象度の PIM を併用することで，WSN 分野における多様なプラットフォームを対象とする移植開発を実現した．また，PIM 間の変換により，異なるプラットフォーム間での移植時に不足しうる設計情報を自動で補完し，低コストでの移植を達成した．

ケーススタディを通し，抽象度の異なるプラットフォーム間の移植を，提案手法を用いることにより，単一の PIM を用いる場合に必要となる手動でのモデル修正コストを排除可能なことを示した．また品質改善工程については新規開発の場合と同様であるが，実機を用いた実験により，本手法における PIM 上での設計変更が，移植先プラットフォームへの適応のための品質改善に実際に貢献することも確認した．すなわち，本開発手法は，

開発対象ごとにWSNソフトウェアの品質を改善する様なソフトウェア開発を支援可能である．

第6章　考察

第 4 章，第 5 章で示したように，本研究は WSN ソフトウェアの新規開発と移植開発における品質改善支援に有用である．第 4 章の結果から，提案手法は新規開発において，要件 1. 新規開発における低コストでのプロトタイプ開発が可能であることを達成することを示した．また，第 5 章の結果から，提案手法は移植開発において要件 2. 移植開発における多様な WSN プラットフォームからの低コストでの移植が可能であることを達成することを示した．更に，第 4 章，第 5 章の結果から，提案手法は新規，移植の双方における品質改善工程にて，要件 3. 通信設計，タスク割当設計を含む細やかな設計変更が可能であること，要件 4. 品質改善のための設計変更を端的に記述可能であることを達成することを示した．すなわち，本研究は開発対象ごとの品質改善支援に求められる要件を全て達成可能である．

しかしながら，本研究では対象とできる WSN ソフトウェア開発や本研究が有効に働く開発に限界が存在する．また，実際には提案した言語や各抽象度での設計は相互に依存している．これは，WSN ソフトウェアでは上位レベルの設計においても観測対象やセンサノードの位置・通信範囲などを考慮する必要があるためである．この様な依存は，言語を分離することの得失を生む．そこで本節では，本研究の有効範囲や限界，言語を分離する場合の得失についての議論，考察について詳述し，本研究の有用性について述べる．

6.1　提案手法の対象範囲

対象とする WSN ソフトウェア

本研究で用いる DSML は，2.1 節で述べた，最も典型的な WSN ソフトウェアをモデル化するために定義したものである．したがって，DataflowML は，イベント通知やイベントハンドラを構成要素として必要とするイベント駆動な WSN ソフトウェアを表現できない．例えば，センサノードをフェンス [104] や活火山 [102] に設置し，一定の加速度を検知した際に侵入や地震というイベントを通知する，イベントが通知されたら全てのノードから情報を収集するなどといった，イベントに基づくアプリケーションは実現できない．ま

たセンサの種類としても，現状では 3.3 節で述べたように TinyOS で既にサポートされている基礎的な物理情報センサのみを対象としている．しかしながら，基礎的なセンサを用いる，最も典型的な WSN ソフトウェアの開発であっても開発対象ごとに品質を高めることは困難であり，本研究はこの部分に対する支援を実現している．将来課題として DSML の拡張が挙げられるが，これについては 8.2 節に詳述する．また，新たなセンサへの対応としては，物理的なセンサとこれに対応する実装，コードテンプレートを個別に用意することで対応可能であると考えている．

対象とする品質改善のための設計変更

品質改善のため，時刻同期アルゴリズムや MAC プロトコルといった，OS やミドルウェア側で実装されている機能に関する設計変更も考えられる．これらの変更は，WSN ソフトウェア単体ではなく，WSN システム全体を開発する場合には実施されうる．一方本研究は，開発者は特定のプラットフォームに向けてアプリケーションソフトウェアを開発することを対象としており，我々の DSML ではこれらの要素に対する設計をモデル上で記述できない．これらの変更は，プラットフォームを管理する技術者が，必要に応じてプラットフォーム側に変更を加えることでしか対応できない．しかし，本研究はアプリケーションレベルの設計を行う場合には有用であるといえる．更に，WSN システム全体の開発という観点では，ハードウェアを含めた協調設計やノード配備を含めたモデル化などが考えられ，幾つかの既存研究が存在する．これらの手法と本研究を組み合わせることは，システム全体の開発に対して有用であると考えられる．

対象とするプラットフォーム

本研究では，開発先として Node-level プラットフォームのみを，移植元として Dataflow-，Group-，Node-level プラットフォームをそれぞれ対象としている．開発先としてはデータ処理，通信，タスク割当の全ての設計を反映可能なよう Node-level プラットフォームのみを対象としているが，Dataflow-，Group-level プラットフォームを対象とした開発も実現可能である．これは，Dataflow-level PIM，Group-level PIM から，各抽象度に属するプラットフォームに対応するモデルへ個別に変換を開発することで達成可能である．例えば Dataflow-level PIM から TikiriDB PSM への変換，TikiriDB PSM から TikiriDB のコード生成をそれぞれ開発，実装する．この様にプラットフォームごとに変換とコード生成器を開発することで，本フレームワークを用いた複数プラットフォームに向けた開発が実現可能である．

また，対象とできるプラットフォームへの制約も存在する．上記の複数プラットフォームへの開発及びプラットフォームからの移植を考える場合，対象プラットフォームが我々の用いた抽象度による分類に属さない場合，本開発手法では対応する PIM を持たないため移植を達成できないと考えられる．これは，本開発手法における PIM が，既存のプラットフォーム分類に基づき定義されているためである．しかしながら，これは既存の WSN 分野における MDA にも同様に存在する限界である．具体的には，エージェントの動作と移動を設計，実装するプラットフォームは通信方式の扱いが大幅に異なるため対象外となる．この様なエージェント型のプラットフォームは，WSN ミドルウェアのサーベイ論文 [98] によれば 15 種中 2 種，WSN 向け DSL のサーベイ論文 [62] によれば 28 種中 4 種である．すなわち，既存研究に基づけば，本研究が対象とできるプラットフォームの範囲は約 8.5 割である．また多段階 PIM の併用により，PSM，PIM 間の乖離が，単一の PIM のみを用いる場合と比べ小さくなる．この結果，PIM2PSM 変換や PSM2PIM 変換の開発コストが低減され，新規プラットフォームに対する高い拡張性が実現できると考えられる．

対象とする WSN の規模

第 4 章，第 5 章のケーススタディの結果から，本研究は中規模程度の WSN に対してもっとも有効であると考えられる．特に NodeML を用いる際に，大規模な WSN を対象とする場合はノードを個別に考慮することは現実的ではなくタスク割当設計が困難である．従って本研究は，特に小規模，中規模の WSN に配備するソフトウェアの品質改善に有用である．しかしながら，NodeML を用いず，GroupML にてマクロな視点から役割の割当条件を設計することで大規模な WSN に対しても品質改善を試みることが可能である．特に既存の自動タスク割当手法を本研究と組み合わせることは，提案手法の有効範囲の拡大につながると考えられる．

対象とする開発工程

本研究は開発プロセス，開発手法に焦点を当てており，品質改善工程における実行結果の監視，施すべき設計変更の分析，設計変更を反映したソフトウェアの再配備などの支援は対象としていない．また，本研究はソフトウェア開発のライフサイクルにおける設計から実装の生成までを支援するものであり，要求の獲得や実装生成後の検証は対象外となっている．これらについては，既存研究との組み合わせにより本研究をより実用に近づけられると考えている．既存研究については第 7 章にて詳述する．

対象とする開発者

開発者の種別として WSN 分野に明るくない初級者や，WSN 分野に精通した熟練者，その間に位置する中級者などが考えられる．本研究で提案した WSN ソフトウェア開発は特定の開発者に向けたものではないが，開発者の種別により有用性に差異が生じることが考えられる．以下では特に初級者と熟練者との違いについて考察，議論する．

初級者にとって，Node-level に属する DSML を用いる WSN ソフトウェア開発は，データ処理，通信，分散処理という異なる関心事が混在するため容易ではない．これに対し，本研究は 3 段階の抽象度で分割した言語として提供しているため，提案 MDD フレームワークを用い混在する関心事を分離可能である．すなわち，初級者は本フレームワークを用いて，まずデータ処理設計のみを Dataflow-level モデルにて行い，続いて自動生成された Group-level モデルにて通信設計を，更に自動生成された Node-level モデルにてタスク割当設計を行う，という段階的詳細化プロセスをたどることが可能である．特に通信設計は，既存の再利用可能な設計を選択的に用いることで，WSN における通信の深い知識を用いず通信設計の変更が可能となる．この際，再利用可能な通信設計は，熟練者などが別途モデル上で利用可能な語彙とコード生成時に用いるテンプレートを事前に用意しておく必要は生じる．この様に本研究は，今後需要が増加する WSN ソフトウェア開発に対する新規参入の障壁を，多段階 MDD フレームワークを用いた段階的詳細化により下げることが可能であると考えられる．

一方で熟練者は，配備先の WSN を考慮に入れ，品質改善を見越した設計を，初期実装獲得工程にてモデルに組み込むことが考えられる．Node-level に属する DSML を用いる場合，新規開発においてはプロトタイプ開発工程にて品質改善のための設計を組み込むことが可能であると考えられる．この場合，4.3 節で示した様に，プロトタイプ開発及び品質改善を同時に実施する場合でも，提案手法は関心事の分離と設計情報の伝播により少ない記述コストでのモデル化を達成可能である．また移植工程においては不足する設計情報を補完する際に最小限の情報のみを補完することが考えられる．これに対しては，5.3 節にて示した様に，提案手法は自動での設計補完により，熟練者であっても移植の度に生じてしまうモデル記述コストの排除に成功している．しかしながら，自動変換により生成されるモデルは，熟練者にとっては冗長な情報を含む．これは，モデルの理解コスト，現実的にはモデル理解の時間の増加につながり，提案手法における不利点であると考えられる．この点については，4.4 節に示すユーザ実験の結果から，WSN ソフトウェア開発の未経験者であっても，品質改善のためのモデル理解と修正を数十分で達成しており，現実的な時間の増加として問題になりにくいと考えられる．

6.2 自動変換における得失

提案手法では，DSML 間の依存を上位から下位のモデルへと伝播させるために自動変換を導入している．しかし，DSML 間の依存関係を考慮せず設計を行う場合，提案手法における自動変換が開発者の意図を下位のモデルへ伝播できないことが起こりうる．

まず，Dataflow-level から Group-level への変換では，Dataflow-level モデルにて決定したデータフローそのものが開発者の意図であり，これを変換後の Group-level モデルへ伝播させる必要がある．この変換では 3.2.1 節にて示したように，Dataflow-level モデルの要素に対し，`DataSink` から下位要素への構造を保ちつつ，一対一の対応で `Group` とそれに付随するリーダ・メンバを生成する．よって，Dataflow-level から Group-level への自動変換ではデータフローは保たれ，開発者の意図はどのような入力モデルに対しても下位のモデルへと伝播する．

Group-level から Node-level への変換では，Group-level モデルにおける通信設計に関するパラメタ，グループ構造の決定が開発者の意図であり，これらを Node-level モデルへ伝播させる必要がある．この変換について，通信設計に関するパラメタは，変換規則にて Node-level の対応するパラメタに直接反映されるため，どのような入力モデルに対しても情報が伝播する．グループクラスタリングの伝播については，まずケーススタディにて，Node-level モデルにおける `Role` の割当が，Group-level モデルにおける，`LeaderNode`/`MemberNodes` の `selectionPattern` を満たしていることを確認し，開発者の意図が正しく伝播していることを確認した．

ただし，各 `selectionPattern` の条件を満たすノードが対象 WSN に存在しない場合には意図通りの変換はできない．データフローやグループ構成の設計は完全に独立ではなく，それを実現する WSN を考慮したタスク割当設計と依存関係にある．このため，WSN のノード構成やタスク割当を考慮せずに Dataflow-level，Group-level の設計をすると，対象 WSN で実現不可能な設計となり変換の失敗が起こりうる．これは，提案手法が記述言語の分離を採用しているために起こる．

このようなタスク割当ができない場合，開発者は，(1) 使用可能なノードでデータフローを再設計，(2) タスク割当の条件を満足するノードの追加，のいずれかを実施する必要がある．(1) を実施する場合は Dataflow-level からの再設計となるため，提案手法では最も手戻りが大きくなる．ただし，データフローの再設計を単一の DSML を用いる手法で実施する場合，通常の開発と同等のプロセスを実施するため 2.5 節で述べた問題が生じる．このため，提案手法において最も手戻りが大きい場合でも，4.3 節で述べた場合と同等の結果が得られると考えられる．(2) はハードウェアやノード配置の設計変更に当たるため本研究の対象外である．ただし，提案手法では変換失敗までにデータフロー設計のみを行っ

ているが，既存手法では各 DSML の抽象度に応じ設計項目が同等或いは増加する．ゆえに変換が失敗する場合でも，4.3 節で述べた場合と同様，提案手法は既存手法に比べ記述能力，記述コストの面で優位と考えられる．

6.3　多段階象度での品質改善における得失

DSML 間の依存関係ゆえに，品質改善工程では複数の抽象度での設計を併用することが考えられる．例えば，データフロー設計とタスク割当設計は独立ではないため，これらの依存関係を考慮し双方を品質改善に併用することも考えられる．この場合，DataflowML-level や Group-level の DSML のみを用いた開発ではノードごとのタスク割当設計ができない．また，Node-level の DSML のみを用いた開発では，データフローとタスク割当の設計，反映を同時に行うこととなり記述が分散，冗長となりうる．一方提案手法では，これらを分離して設計，記述することができ，適切な抽象度にて個々の関心事に焦点を当てることができる．設計変更は上位から下位へと段階的に行うことで，自動変換が設計を伝播させることが可能である．

ただし，提案手法では言語を分離しているがため，言語間の依存関係を無視した設計が生じうる．この結果，前述の変換失敗の他，モデル間の非一貫性問題，自動変換による設計の上書き問題が起こりうる．

提案手法では，3 つの DSML でモデルを記述し，開発の中で任意のモデルに対して変更を加えていく．この際，開発中に保持するモデルの間で一貫性が崩れることが考えられる．非一貫性の問題として，例えば対象 WSN ではどのようにタスク割当を設計しても実現できないデータフローを設計してしまう場合が挙げられる．或いは，開発のある段階で GroupML にて計測間隔の値を変更した場合，変更を反映した Node-level モデルと実装は自動生成されるが，Dataflow-level モデルとの間で値が異なる，という矛盾が生じることも考えられる．品質改善工程を何度も繰り返す場合においては，この様な成果物の間の矛盾は大きな問題となる．

また，設計の上書き問題として，下位のモデルを変更した後に上位のモデルを変更すると，自動変換により下位のモデルでの変更の上書きが起こりうる．これは，提案するモデル変換が抽象度の高いモデルから低いモデルへの単方向のみを対象としているためである．

現状では，これらの問題は開発者が一貫性を保つよう，或いは変更が上書きにより失われないよう注意深くモデルを管理する必要がある．これらに対し将来課題として，双方向変換技術を用い，下位から上位へも変更を伝播させることで開発者の負担を低減できると

考えている．

6.4 その他の得失

本節では変換時，品質改善時以外において，記述言語の分割がどのような得失をもたらすかを議論する．まず異なる関心事を同時に扱う方法として，提案手法のように抽象度ごとに記述を分割する他，抽象度の高い言語を一つ設定し，カスタマイズ可能な部分をパラメタとして設定可能にする手法が考えられる．このような可変部分を含めた単一の言語を用いる場合，前述の非一貫性や設計の上書きの問題を生じずWSNソフトウェアの設計を行うことができる．

しかしパラメタによるカスタマイズでは，データフローやグループクラスタリングの変更のような，要素の追加，削除，つなぎ変えの組み合わせからなる変更を扱うことは難しい．一方，提案手法では抽象度ごとにDSMLを定義し記述を分割することで，非一貫性の問題は生じうるものの，パラメタによる設計変更だけでなく，パラメタの変更では表現することが難しい設計変更も可能としている．

その他にも，言語を分離することでモデルの再利用性向上，関心事の分離が促されると考えられる．ネットワーク非依存なDataflowML，ネットワーク依存かつノード非依存なGroupML，ネットワーク及びノード依存なNodeMLのように，3つの抽象度を明示的に分割する事で記述されたモデルの非依存な部分に対する再利用性が高まると考えられる．更に依存，非依存が明示的に分割されることから，開発者が対象とするソフトウェア成果物が移植可能か否かを判定する際の成果物の可読性の向上が期待される．例えば，DataflowMLで記述したモデルはネットワーク非依存であり，同一のデータフローを実現する場合は異なるWSNに対してもそのままモデルを再利用可能であり，データフローのみに焦点を当てているため，抽象度の低いモデルやコードと比較し理解容易性が高いといえる．また，DataflowMLを用いる際はネットワークやノードの知識を要さないため，ドメインの専門家でもモデルを記述，理解しやすく，ドメイン専門家とWSN専門家との間で関心事の分離がより明確になるという利点がある．

次に学習コストについて，提案手法は単一のDSMLを用いる開発方法よりも多くの学習コストを要する．提案手法では3つのDSMLを用いることを前提としており，単一のDSMLを用いる場合と比べ言語の学習コストが高くなる．しかしながら，3つのDSMLを用いることで，Dataflow-levelやGroup-levelのDSMLのみを用いた場合には記述できない設計が記述可能となり，Node-levelのDSMLのみを用いた場合よりも低いコストでモデルを記述可能となる．したがって，一度3つのDSMLを学習しさえすれば，単一の言

語を用いる場合よりも低いコストで品質を改善した WSN ソフトウェアが開発可能になるといえる．また，ユーザ実験においても，現実の開発者が数十分の DSML の説明によりモデルを記述可能であり，品質改善の支援に有用であると考えられる．

第7章　関連研究

7.1　WSNソフトウェアの品質改善支援に関する研究

WSN ソフトウェアにおいては，品質への要求がアプリケーションごとに異なり，また品質自体はソフトウェアが実行される環境に依存する．したがって，WSN ソフトウェアを，対象とするアプリケーション及び環境ごとに適応させ，品質を改善することは重要課題である．WSN ソフトウェアの品質改善は，2.2 節で述べたように，初期実装を獲得し，実行，監視，分析，修正を繰り返すことで達成される．したがって，本節では WSN ソフトウェアの品質改善における実行，監視，分析，修正の各工程に対する支援技術について述べ，本研究の位置付けを示す．初期実装獲得についての関連研究は，ソフトウェア成果物に対する直接の編集を行うという観点で修正の工程と同じであるため 7.1.2 節にて併せて述べる．

7.1.1　実行，監視，分析の支援

実行の支援

品質改善における実行では，特に修正が施されたソフトウェアの再実行のコストが問題となる．ソフトウェアの修正から新たに得られた実装は，各センサノードに再度配備しなければならず，品質改善のたびに手間が生じる．WSN では無線通信が可能なため，新たなソフトウェアを，無線通信を通し配備することで人手による手間を削減可能である [99] しかしながら，再配備の都度，修正されたソフトウェア全体を WSN 全体へ配備することは通信量に対する負担の増加を招く．このような再配備の問題を解決するため，変更部分を検知し，再配備における通信量低減を試みる研究 (Reprogramming) がなされている．

Reprogramming の手法は，大きく分けて仮想マシンを用いる手法と専用のプロトコルを用いる手法に大別できる．仮想マシンを用いる手法としては Mate [50] などが挙げられる．Mate では，各センサノードに仮想マシンを配置し，仮想マシン上で動作するコードを配備する．仮想マシン上で動作するコードは，通常の OS 上で動作するコードに比べて

小さいため，少ない通信量でコードの更新が実現可能である．しかしながら，仮想マシンを用いる手法はコードの実装環境と深く結びついており，プラットフォーム依存な手法となる．

また，専用のプロトコルを用いる手法としては，Zephyr [67] がある．Zephyr では，更新前と更新後のバイトコードを比較し，この比較結果に対して Rsync のアルゴリズムを適用する．これにより，更新分だけを波及させるため少ない通信量でのコード更新を実現している．また，バイトコードの比較を行うためプログラミング言語や OS，ミドルウェアといったプラットフォームに非依存な手法となっている．特に後者の手法は特定のプラットフォームに依存しないため，本研究と組み合わせることは有用であると考えられる．

監視の支援

品質改善における監視では，WSN ソフトウェアの実行時の品質，すなわち実行時性能を得ることが主たる目的となる．実行時性能を取得する方法として，実際に開発対象とする環境にて WSN ソフトウェアを実行し，その性能を測定する方法と，シミュレーション実行により推定された性能を測定する方法の 2 つが考えられる．

WSN 用のモニタリングツールとしては，WSN ソフトウェアに測定用データを送る機能を組み込むことで能動的に情報収集を行うものと，ソフトウェアを実行する WSN とは別の監視用 WSN を用い受動的に情報収集を行うもの 2 種類が存在する．能動的なモニタリングツールとしては，MOTE-VIEW [94]，SeeMote [83] が挙げられる．これらのツールでは，情報収集のためのプロトコルを監視対象とする WSN ソフトウェアに導入し，性能測定に必要な情報を得る．MOTE-VIEW はホストコンピュータ上で動作するツールも提供しており，各ノードの計測値，ノードのもつプロファイル，ホストに蓄えた計測データへのアクセスを実現すると共に，Lifetime の推定や送信パケットの到達率などを測定可能である．SeeMote は，センサノードに取り付け，ノードの計測データやエネルギー消費量を表示可能なディスプレイを含むハードウェアと，ディスプレイに情報表示を行うソフトウェアを提供している．

受動的なモニタリングツールとしては PMSW [106] がある．PMSW では，要求を達成するソフトウェアを実行する WSN とは別に，監視用の WSN を用意し，このネットワークを通して実行時性能の測定に必要な情報を取得する．監視対象とする WSN におけるパケットやその損失率，ネットワークトポロジなどの情報を，監視対象から独立した WSN で取得することで，監視対象に非依存なツールを実現している．実世界での実行による性能取得は正確であるものの，その都度実環境で実行しなければならず，またモニタリングツールの導入にもコストを要する．

また，WSN ソフトウェアの実行時性能を，シミュレータを用いることにより推定することも可能である．WSN 向けのシミュレータに関する研究は数多く存在し [90]，代表例としては TOSSIM [51]，SensorSim [69] などが挙げられる．TOSSIM は TinyOS 用のネットワークシミュレータであり，NesC のコードを実行しアプリケーションの性能を推定可能である．SensorSim はネットワークシミュレータ ns-2 [1] の拡張であり，電力モデルや通信チャネルを組み込むことで WSN のシミュレーション実行を実現している．また，仮想空間内に WSN を配備することで，障害物の影響などを含む正確な性能推定を目的とするシミュレータ，WonderSim [9] も存在する．

これらのシミュレータは，実際にプログラムを WSN に配備することなく性能を測定できるため利用の敷居が低く，設計の妥当性確認に用いることができる．しかしながら，シミュレーションにあたりある程度現実を捨象したモデルを用いるため，測定精度はシミュレーションモデルに依存する．多くのシミュレータは，WSN ソフトウェアが対象とする環境の物理的な要素を考慮していないため，障害物などにより生じる実行時性能の変化を反映することはできない．また，WonderSim のような物理環境を含むシミュレータを用いる場合でも，実世界に存在するノイズ源などまでモデル化することは難しく，また対象とする環境を仮想空間としてモデル化する必要があり，対象環境ごとにモデルを作成する手間が生じる．

モニタリングツールとシミュレータは，WSN ソフトウェアの実行時性能の取得手段として重要であり，それぞれに一長一短が存在する．従って，実環境での実行前の妥当性確認としてシミュレータを用い，更に正確な実行時性能の取得のためにモニタリングツールを用いるという形で本開発プロセスと組合せて用いることは有用であると考えられる．

分析の支援

品質改善における分析では，WSN ソフトウェアの品質を改善するための適切な設計変更を決定する．設計変更を決定する方法は，開発対象への要求や既存設計のトレードオフを考慮しなければならないため人手での意思決定により実施される．

人手での意思決定による設計変更については，既存設計の得失の分析による支援が考えられる．既存設計の得失を開発者に示すことで，対象とする開発における適切な設計の選択を促すことが可能である．例えば，通信設計の一種であるルーティングプロトコルには，プロトコルごとに得失が存在 [52] し，適するアプリケーションが異なる [96] ことが知られている．Shrestha ら [85] は Mesh, Clustered，Tree の 3 種の，Mamun [58] は Flat，Cluster-based，Tree-based，Chain-based の 4 種のネットワークトポロジの分析を実施し，

[1] http://www.isi.edu/nsnam/ns/

それぞれの利点と欠点を論じている．Pantazis らはサーベイ論文にて既存の WSN 用の通信プロトコルを調査，分析，分類し，それらが持つ得失について示している [68]．また Upadhyay らは，特に実行中に通信経路を選択的に用いるアドホックなプロトコルに着目し，その得失と適するアプリケーションの種類について論じている [96] その他，WSN における圧縮手法についての比較調査結果 [73] なども同様に適切な設計選択の促進に利用可能である．これらの結果を用いることで適切な設計の選択支援が可能であると考えられ，これらは本開発プロセスにおける品質改善工程においても有効であると考えられる．

7.1.2　修正の支援

初期実装の獲得や実行，監視，分析の結果として決定された品質改善のための設計変更は，DSL を用いプログラムを直接修正するか，WSN ソフトウェアを実装に非依存な表現したモデルを修正するかのいずれかにより達成される．本節では，WSN ソフトウェア開発全体や WSN ソフトウェアの品質改善を支援する技術を，DSL を用いた支援と，MDD の技術を用いた支援とに大別し示す．

DSL による支援

DSL に関しては，2.3 節で述べたように，DSL の実行環境であるプラットフォームと併せての提案が数多くなされている．また，これらは幾つかのサーベイ論文 [89, 62] によりまとめ，分類がなされている．WSN 向け DSL は，Mottola らのサーベイ [62] では 28 種類，Sugihara らのサーベイ [89] では 36 種類の言語が取り上げられている．いずれのサーベイ論文にも共通する分類の視点として，プログラムのもつ命令が実行されるスコープ，すなわち DSL が開発者に与える抽象度がある．この視点に基づくと，WSN 向け DSL は以下の 3 つに分類できる．

分類 1. 開発者が記述したプログラムの命令が，個々のノード上で実行される DSL

分類 2. 開発者が記述したプログラムの命令が，特定のノード集合に属するノード上で一斉に実行される DSL

分類 3. 開発者が記述したプログラムの命令が，WSN 内の全てのノード上で実行される DSL

本研究ではこれらの分類をそれぞれ Node-level，Group-level，Dataflow-level と呼んでいる．これらの DSL は，特定の関心事を捨象し，開発者に対して隠蔽することで WSN ソ

フトウェア開発の難しさを低減している．

Node-level に属する DSL の代表としては NesC や TeenyLIME が挙げられる．これらの DSL のプログラムはセンサノードのハードウェアの差異を捨象し，個々のノードの動作を記述するものであり，プログラム内の命令はプログラムが実行されるノードにのみ影響する．このような DSL では，通信やタスク割当に関する細やかな設計変更による品質改善が可能であるものの，データ処理と通信，タスク割当の関心事が混在し，プロトタイプを得るまでの設計・実装が複雑となる．

Group-level に属する DSL としては Abstract Regions や Regiment がある．これらの DSL では個々の分散ノードの概念を捨象しており，開発者がノード集合を形成する条件 (e.g. N-ホップ圏内，距離 d 以内，等) を定義し，このノード集合が担う計測，集約等の処理をプログラムとして記述する．すなわち、記述されたプログラムは，開発者が定義したノード集合にのみ影響する．このようにマクロな視点での通信にのみ着目することで，Group-level に属する DSL では既存の通信設計の再利用が促進されている．しかしながら，タスク割当については変更出来ず，またプロトタイプ開発もデータ処理と通信設計を同時に扱うため複雑となる．

最後に，Dataflow-level に属する DSL として TinyDB や TikiriDB が挙げられる．これらの DSL は WSN 内での通信に関する概念を捨象しており，記述されたプログラムは WSN 内の全てのノードの動作を表す．これにより，開発者はデータ処理のみに焦点を当てることが可能となる．しかしながら，通信に関する設計は，DSL に対応するミドルウェア側で実装された固定的な設計を用いることになる．たとえば，TinyDB のミドルウェアはセンサ値の収集にツリー型のネットワークを用いた通信を固定的に用いている．この結果，これらの DSL では通信に関する設計は変更ができず，WSN ソフトウェアの品質改善が限定されてしまう．

このように数多くの DSL が提案されているが，2.3 節で述べたように，単一の DSL，プラットフォームを用いた開発では，WSN ソフトウェアの品質改善のための要件を全て同時に満たすことが出来ない．これに対し本研究では，多段階抽象度を併用する MDD フレームワークを用い，全ての要件を達成する開発プロセス，開発手法を実現した．

モデル駆動開発による支援

幾つかの研究では，MDD の WSN ソフトウェア開発への適用を試みており，サーベイ論文 [57] でも 16 の研究が挙げられている．これらの研究は 2.4 節で述べたように，MDD にて用いる DSML の抽象度により分類可能である．多くの既存研究では，WSN ソフトウェアを，モデルを用いて抽象化することで実装コストの低減を試みている

個々のノードのタスクや動作を記述する Node-level に当たる DSML を用いる手法として GRATIS, Thang ら，Beckmann ら，Mozumdar ら，Wada らの手法がある．GRATIS [?] ではモデルからのコード生成による実装コストの低減を実現している．GRATIS は NesC コンポーネントの依存関係や合成を直接モデル化するグラフィカルなツールを提供しており，記述されたモデルはツールを通してコードへと自動で変換される．

Thang らの手法 [91] では，個々のノードに割当てられるセンサや通信インターフェース，タスクを表現する Node-level DSML を提案している．作成されたモデルはコード生成器を通して NesC のコードへと自動変換される．これにより，モデル上では通信，タスク割当の設計を表現しつつ，実装にかかるコストの低減を実現している．

Beckmann らの手法 [10] では，Object Management Group によって管理・標準化されているミドルウェアである DDS(Data Distribution Service for Real-time System) 上で動作する WSN ソフトウェアを開発するための MDD 手法を提案している．提案する開発手法では，開発者は WSN ソフトウェアが必要とするデータの種類，各ノードに与える役割，ノードが持つリソース制限など，個々のノードに関するモデルの記述を行う．記述されたモデルは DDS 上で実行可能なコードへと変換される．

Mozumdar らの研究 [63] では，WSN ソフトウェアを複数のプラットフォームに向けて開発するためのフレームワークが提案されている．このフレームワークでは，開発者は WSN ソフトウェアを個々のノードの振舞として Simulink を用いてモデル化している．すなわち，我々の分類における Node-level に属するモデルを用いている．記述されたモデルはコード生成器を通して TinyOS，MANTIS の 2 つの OS 上で動作するコードへと個別に詳細化可能である．記述されたモデルはシミュレーション実行も可能としている

また，Wada らの研究 [97] では，アプリケーションの実行環境として，BiSNET [14] と呼ばれる，WSN ソフトウェアを実行するモバイルエージェントと，モバイルエージェントが動作するミドルウェアを対象とし，この実行環境で動作する WSN ソフトウェアを，UML を用いてモデル化する手法を提案している．提案されたモデル化手法では，開発者はエージェントの構造と振舞をクラス図とシーケンス図を用いて，ネットワークの構造を，オブジェクト図を用いてそれぞれ記述する記述されたモデルはそのまま解釈，実行されるため実装コストの低減に成功している．

ノード集合の動作を記述する Group-level に当たる DSML を用いる手法として，Losilla ら，Rodrigues の手法，LWiSSy，Baobab がある．Losilla らの研究 [54] では，特定のミドルウェアや OS に依存しない WSN 用のモデリング言語を定義している．提案されたモデリング言語ではノード集合の振舞が記述可能でありこれは我々の分類における Group-level に属する DSML である．記述されたモデルはモデル変換とコード生成を通し NesC に特化したモデル，NesC コードへと詳細化可能である．Losilla らによれば，このモデリング言

語は，変換規則を作成すればNesC だけでなく他の言語にも変換可能であり，ミドルウェアや OS に依存しないものである．

Rodrigues らの研究 [75] では，DSML で記述されたモデルを PIM としている．また，PSM は PIM から自動生成され，WSN 専門家が品質改善のために手動で修正するモデルとしている．この際，PSM は特定の OS(TinyOS と Squawk VM) に依存するモデルであり，品質改善はプラットフォーム依存の形で実施される．ArchWiSeN [74] は，上記の Rodrigues らの研究に振舞を表すモデルを追加することでより詳細な設計を実現している．WSN ソフトウェアは構造，振舞の両側面から PIM として記述され，品質改善が可能な PSM へと自動変換される．開発者は構造，振舞の両側面から，PSM に設計変更を施すことで品質改善を試みる．LWiSSy [23] では，Losilla らの DSML を拡張し，構造，振舞，最適化の 3 つの視点から設計された DSML を提供している．LWiSSy は同じ抽象度にて複数の視点から WSN ソフトウェアを記述することで，より詳細な設計を実現している．これらの DSML は PIM を記述するために定義されており，記述されたモデルは Rodrigues らの研究と同様に PSM に変換され品質改善がなされる．

Akbal-Delibas らの研究 [3] では，種々のアプリケーションドメインへの適用を考慮した WSN 向け MDD にフレームワーク Baobab を提案している．Baobab 上では，適用先ドメインに非依存なメタモデル (GMM: Generic Meta Model) が定義されており，開発者は GMM を開発対象とするドメインとプラットフォームに合わせたメタモデル (DSMM: Domain-Specific Meta Model と PSMM: Platform-Specific Meta Model) へと拡張する．この拡張されたメタモデルに基づき，開発者は対象とする WSN ソフトウェアをモデル化，開発する．GMM は同種のタスクを担うノード集合の動作を表現するよう定義されており，これらの拡張である DSMM や PSMM に基づき記述されたモデルは Group-level に属するモデルとなる．記述されたモデルは前述の研究と同様に実行可能なコードに自動変換可能となっている．

WSN 全体の動作を記述する Dataflow-level に当たる DSML を用いる手法は少なく，DFuse や SM4RCD が挙げられる．DFuse [48] では，開発者は WSN ソフトウェアに必要なデータ処理のみを記述し，フレームワークがこの処理をタスクに分割，自動で各ノードへと割り当てる．これによりデータ処理のみに焦点を当てた開発を実現している．SM4RCD [35] は，サービス指向アーキテクチャの考えに従い，WSN ソフトウェアを互いに疎なサービスの組み合わせにより構築することを目指している．まず Glombitza らはセンサノード上でサービスを実行するための基盤を提案し，この上で実行するソフトウェアを UML の状態遷移図を用いて表現する手法を提案している．これにより，開発者は WSN ソフトウェアを，どのようなサービスの組み合わせにより実現するかのみに焦点を当てて開発することができる．

上述の手法は，モデルの直接的な実行やコードの自動生成により，実装にかかるコスト低減し，生産性を向上させることに成功している．しかしながら，開発者が扱う DSML は単一の抽象度に限定されており，WSN ソフトウェアの品質改善においては，2.5 節で述べた品質改善のための要件を全て同時に満たせない．また，上述の幾つかの手法では，開発コストの低減の他，MDA の概念を用いたソフトウェア移植に焦点を当てている．Mozumdar らの研究，Losilla らの研究，Rodrigues らの研究では，DSML をプラットフォーム非依存となるよう定義し，これらを PIM として移植を達成している．しかしながら，いずれの手法も WSN 向け OS(Node-level プラットフォーム) 間の移植を対象としており，WSN 分野における多様なプラットフォームは考慮していない．これらの手法では単一の抽象度の PIM のみを用いており，2.5 節で述べた品質改善のための要件を全て同時に満たせない．これに対し本研究では，複数の異なる抽象度を併用する MDD フレームワークを提案することで全ての要件を同時に満たしている．

その他，WSN へ MDD を適用する手法として，WSN ソフトウェア開発だけでなく，要求の獲得やハードウェア，ノード配備も含めた WSN システム全体の開発を対象とした手法も存在する．Paulon らは，WSN ソフトウェアが持つ機能性を網羅的に表現するフィーチャモデルと，選択されたフィーチャに対応するよう定義された WSN ソフトウェアの構造を表現する UML プロファイル (WiSeN) を提案している [71]．これにより，明示的な WSN ソフトウェアへの要求の獲得と，要求と WSN ソフトウェアの設計都のトレーサビリティ確保を図っている．また Doddapaneni ら [26] や Ebeid ら [29] は，個々のノードの動作を表すモデルに加え，WSN を構成するノードの配置を表すモデルも提案，併用している．これにより，物理環境を可視化した上でのタスク割当設計をモデル上で表現しており，記述されたモデルは物理環境を含めてシミュレーション実行可能としている．Abrishambaf らはハードウェアの選択，ハードウェアへのソフトウェアの割当までを含めた，設計から実装までの支援を，UML プロファイルを用いて実現している [1]

本研究は WSN ソフトウェア開発の支援を対象としており，要求の獲得やハードウェア，ノード配備の決定は対象外としている．しかしながら，実際のソフトウェア開発においてはこれらの活動も不可欠であり，本研究とこれらの研究を組み合わせることは，WSN ソフトウェア開発のライフサイクル全体を支援するという観点で非常に有用であると考えられる．

7.2 WSN 分野以外でのモデル駆動開発適用に関する研究

複雑化するシステム及びそれに付随するソフトウェア開発のため，モデルを開発における成果物の中心に据える MDD [81] 技術が研究，整備されてきた．MDD において，開発者はソフトウェアを実装よりも抽象度の高いモデルとして記述し，変換を通して実装を獲得する．モデルを用いた抽象化により，問題領域と解決領域との乖離を埋めることでソフトウェア開発の品質及び生産性の向上を図っている [33]．問題の本質をモデルとして切り出し，実装詳細など他の関心事を捨象することは，特定の関心事にかかる設計への注力を促すため品質向上につながる．モデルからのコード生成は，実装にかかるコストを削減し生産性向上につながっている．産業界での適用事例においても MDD の品質，生産性に対する効果は報告がなされている [44] [60] [61] [103]．

しかし，この問題領域と解決領域との乖離を単段階の変換 (コード生成) のみを用いて埋めることは困難である．単一モデルのみを用いる MDD(図 7.1(a)，以下，単段階 MDD) では，記述したモデルをコード生成器に入力し実装を獲得する．この際，コード生成器はモデル上の情報に加え，コード生成に必要な情報を固定値として付与することで実装を出力する．

記述するモデルの抽象度が高い場合，モデル記述は簡潔に済むが，コード生成器が付与する情報量が増す．コード生成器が与える情報はソフトウェア品質に影響を及ぼす設計上の決定を含みうるが，生成器は固定された決定しか与えない．このため，高抽象度のモデルを用いる場合には詳細段階での設計が行えず，設計上の柔軟性が失われ，品質の確保が困難となる．

或いは，モデルの抽象度が低い場合は，コード生成に必要な情報のほとんどをモデル上に記述する事となる．詳細なモデルの記述は，適切な分析と設計なしに実装を行うことと同様に困難である [77]．更に，この様なトレードオフのバランスを取るような適切な抽象度を設定することもまた困難である [95]．

この様な乖離を多段階の変換により埋めることは，ソフトウェア工学においては広く用いられる．80 年代には多重プログラミングシステムの設計において，物理層とユーザ層との乖離を埋めるために多段階抽象度が導入された [25]．また，問題に紐づく抽象度と，実装言語が提供する抽象度との乖離を埋めるため，手動での段階的詳細化の概念が導入された [45]．更にソフトウェア開発プロセスでも，要求，分析，設計，実装の工程を経て，主に手動で段階的に詳細化する手法が用いられる [47] [76]．したがって，MDD において問題領域と解決領域の乖離を，多段階の自動化された変換で埋めることは有効であると考えられる．これまでの多段階変換の概念は手動が前提であったが，MDD においてはこれらの変換を自動で行う．

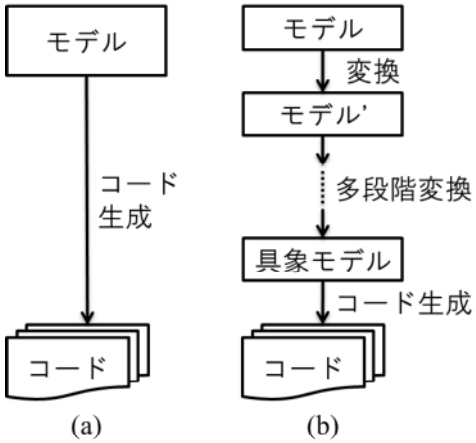


図 7.1: 単段階 MDD プロセスと多段階 MDD プロセス

段階的詳細化を適用し，多段階の変換を用いる MDD(以下，多段階 MDD) を図 7.1(b) に示す．多段階 MDD では，コード生成に用いる低抽象度のモデル (以下，具象モデル) の記述に必要な設計を，高抽象度のモデルから段階的に決定する．複数のモデルを用い，自動変換で上位の設計を下位へ伝播させることで，各抽象度での設計上の関心事が絞り込まれるため，多段階 MDD では以下の 2 点が期待される．多段階抽象度で複数のモデルを用いることは，単段階 MDD に比べ，各抽象度におけるモデルの関心事が更に絞られることにつながる．これにより，品質面では関心事を絞った検証や性能推定による品質保証が期待される．また生産性の面では，高抽象度での決定を低抽象度に伝播させるモデル変換を用いることと，関心事の絞り込みにより簡潔なモデル記述が可能となり生産性向上が期待される．

幾つかの研究では，多段階 MDD を特定の分野へ適用した事例を報告している．ドメイン特化型の MDD はその有用性が見込まれており [108]，多段階 MDD が適する分野としては分散システムが挙げられる．具体例としては通信分野 [31]，Automotive システム [79] [59]，SoC[72] などでの適用事例が存在する．また MDA による移植支援としても，エージェントシステム [38] やコンテキストアウェアシステム [7]，サービス指向アーキテクチャ [53]，クラウドアプリケーション [6] など幅広い分野に適用されている．例えば [59] は Automotive システムにおいて，機能性を表現するフィーチャモデル，機能性の実現方法を表現するデータフローコンポーネントモデル，ハードウェア上でのソフトウェア配置を表現するネットワークモデルへの段階的詳細化による開発を提案している．

しかし，これらの多段階 MDD の適用事例では性能推定などによる品質面への効果を評価しており，多段階 MDD において，簡潔なモデル記述による生産性向上を評価した調査

は我々の知る限り存在しない．これまで行われてきた MDD の有用性評価では，主にモデル上で検証や性能推定による品質向上と，コード生成による生産性向上の 2 つに焦点が当てられてきた．また，産業界での適用事例 [44] [60] [61] においては，生産性の観点では主にコード生成によるコスト削減を評価しており，具象モデルを獲得するまでのコストについての言及はない．更に，特定分野へ多段階 MDD を適用する研究では，関心事の分離により可能となる，特定環境での性能推定の正確さといった品質面に焦点を当てた評価のみを行っている．このように，現状では多段階変換を用いる事による，具象モデル獲得にかかるモデル記述コストの低減についての調査が欠けており，多段階 MDD の有用性が正しく評価されていない．すなわち，多段階 MDD では，複数のモデルと自動変換によるモデル記述コストの低減が期待されるが，現状ではこの点についての評価が欠けている．

これに対し本研究では，多段階 MDD の生産性に対する有用性を WSN 分野にて評価した結果を示している．特に，多段階 MDD プロセスと単段階 MDD プロセスとの比較を，具象モデルを獲得するまでに開発者が行うモデル記述コストの観点から行っている．ケーススタディの結果は，多段階変換を用いることが，開発者が行うモデル記述のコストを低減すること，すなわち生産性への貢献を示している．

第8章　結論

8.1　本論文のまとめ

無線通信，バッテリ駆動という特性からくる敷設の容易さから，WSN は実世界と連動するシステムの情報観測基盤として重要な役割を果たすためその需要が高まっている．WSN を実現するソフトウェアは実行する環境により品質が変化するため，その開発は開発対象ごとに品質を高める必要があり手間が大きい．この様な開発対象ごとの品質改善を達成するためには，低コストで初期実装を獲得すること，品質改善のための設計を端的にかつ細やかに変更できることが求められる．しかしながら，既存の WSN ソフトウェア開発手法ではこれらの要件を全て同時に満たすことが困難である．これは，既存手法がいずれも単一の抽象度のモデルを対象とした際に生じるトレードオフを持つためであったそこで本研究では，複数の抽象度を併用する多段階 MDD フレームワークを提案し，これらを新規開発，移植開発の双方へ適用した．

第 2 章では，WSN ソフトウェア開発に於ける，開発対象ごとの品質改善への要件と，既存手法における問題を詳述した．まず本論文が対象とする WSN ソフトウェアの種類を示し，品質改善を実現するための要件を具体例と共に示した．初期実装獲得工程においては，新規開発では低コストでのプロトタイプ開発が，移植開発では多様なプラットフォームからの低コストでの移植がそれぞれ求められる．品質改善工程においては，品質改善のための細やかな設計変更と，設計変更の端的な記述がそれぞれ求められる．また，既存研究として，WSN 向けプラットフォームを用いた開発，MDD を適用した開発の現状についてそれぞれ述べた．その上で，本論文が取り組む WSN ソフトウェアの開発対象ごとの品質改善における課題を詳述している．既存手法はいずれも単一の抽象度のみに焦点を当てており，抽象度間のトレードオフの関係から，これらの手法では上述の要件をすべて満たすことが困難であることを示した。

これを受け，続く第 3 章では，前述の問題を解決するために用いる多段階モデル駆動開発フレームワークを提案した．単一の抽象度ではなく，複数の異なる抽象度を併用，使い分けるため，提案フレームワークで併用する 3 種の DSML を提案した．また，複数のモデルを併用する際に問題となるモデル間の一貫性について，自動モデル変換を提案する

ことにより単方向の一貫性を確保した．これらのフレームワークはコード生成器を併せ Eclipse 上でプラグインとして実装し，この実装詳細についても詳述した．

第 4 章，第 5 章では，それぞれ提案フレームワークを用いた WSN ソフトウェア開発手法を提案している．第 4 章は，提案フレームワークの新規開発への適用を提案した．提案したプロセスは，プロトタイプ開発工程ではデータフロー設計のみを，品質改善工程ではデータフロー設計，通信設計，タスク割当設計の全てを扱うよう定義し，複数の抽象度の使い分けを提案した．このプロセスについて，実世界での運用実績のある WSN ソフトウェアを開発するケーススタディを実施し，最も実装に近い DSML のみを用いる場合と比べ，3 つの DSML を併用することでモデルの記述コストが低減しつつ，通信，タスク割当の設計変更も反映可能であることを示した．更に，ユーザ試験評価から，現実の開発者が，本プロセスを用いることにより WSN ソフトウェアの開発と品質改善が可能であることも示した．

第 5 章では，提案フレームワークの移植開発への適用を提案した．3 つの DSML で記述されたモデルを PIM とみなし，3 つの PIM を併用する MDA を構築することで，単一の PIM のみを用いる場合に比べより多様なプラットフォームからの移植開発を可能とした．提案する開発手法を用い，抽象度の異なるプラットフォーム間の移植開発を行うケーススタディを通し，本開発手法が低コストでの移植と，細やかで端的な設計変更を達成することを示した．更に実機を用いた実験により，本手法における設計変更が，実際の品質改善に寄与することも確認した．

第 6 章では第 4 章，第 5 章の結果を受け，本研究の有用性や限界について述べた．本研究の特徴である 3 段階の抽象度の DSML の併用による得失について，一貫性，学習時間などの観点から議論した結果を記した．また，提案手法の適用範囲について議論し，本研究の限界を示した．

第 7 章では関連研究について論じた．まず，WSN ソフトウェアの品質改善支援について，実行，監視，分析，修正の観点から関連研究を示し，本研究の WSN 分野における位置づけを論じた．特に WSN ソフトウェア開発に直接関わる修正については，DSL を用いた研究，MDD を用いた研究との比較を示した．WSN 以外の分野における MDD の適用に対する関連研究についても述べ，本研究の MDD 分野における位置づけについても論じた．

8.2 今後の展望

本研究では多段階 MDD を用いた WSN ソフトウェア開発手法を提案しその適用を行った．しかしながら，より実用的，効果的な開発の支援に向けて以下で述べる拡張が考えられる．

8.2.1 対象ソフトウェアの拡張

本研究における DSML は，最も一般的な WSN ソフトウェアである，計測のみを目的とし，移動性を考慮せず，データの計測・送信を定期的に行うものを対象として設計した．しかしながら，今後の WSN の普及とともに，より多様で複雑な WSN ソフトウェアの興隆が考えられる．したがって，本研究の今後の拡張として，提案した DSML を，アクチュエータを含めた Sense and React 型の目的を持つ WSN ソフトウェアやノードが移動性を持ちうる WSN ソフトウェア，データの計測・送信をイベント駆動に実行する WSN ソフトウェアを記述できるよう拡張することが考えられる．拡張のためには，アクチュエータや移動性，イベント駆動を記述可能な DSL や，既存の WSN ソフトウェアを分析し，ボトムアップに記述に必要な要素を同定していくことが考えられる．

また，本研究の DSML は WSN ソフトウェアの静的な構造面をモデル化しており，動的な振舞面はモデル化していない．現状では，振舞に関する情報は，コード生成時のテンプレートに埋め込まれており，開発者が明示的に設計するための支援は行っていない．より柔軟に WSN ソフトウェア設計するため，振舞をモデル化する既存研究を基に，各抽象度に対し動的側面をモデル化する DSML を定義し，モデル間をつなぐ変換を本研究と同様に定義することで振舞面に関する支援が可能であると考えている．また，振舞面に対するモデルを導入する場合，抽象度間の一貫性だけでなく，同抽象度の構造面に対するモデルとの一貫性も考慮する必要がある．このような一貫性保持については，OCL 記述を用いた非一貫性の検知やモデル同期の技術を導入し解決を図ることが可能であると考えている．

8.2.2 WSN システム全体の開発への拡張

本研究は WSN ソフトウェアの開発支援を目的としたが，WSN ソフトウェアはその実行環境である OS やミドルウェア，ハードウェア，配備形式と相互に関連している．例えば，第 1 章でも述べた様に，WSN システム全体の品質は，ソフトウェア以外の要素の設計変更によっても改善される．より柔軟で品質を改善しやすい開発のためは，OS やミド

ルウェア層での設計変更を含むモデル，ハードウェアとの協調設計，WSN が実行される環境と配備状況のモデル化を行う研究と本研究を組み合わせることが考えられる．また，要求の獲得，要求と設計，実装とのトレーサビリティ確保により，より網羅的な開発支援が可能となる．更には実装生成後の検証についても，実環境での実行とともにシミュレーション手法との組合せによる支援も考えられる．7.1 節で述べたように，これらの工程に対し，既存研究との組み合わせにより対象範囲を広げる事でソフトウェア開発におけるライフサイクル全体を支援するより実用的な開発の支援が可能になると考えている．

8.2.3　品質改善のための設計の決定工程の支援

提案手法での品質改善工程において，開発者は現状の品質を分析し，既存の再利用可能な設計の中から品質改善のための設計を選択する．この分析と決定の工程については，現状では開発者に依存する形をとっている．すなわちこの工程では，モデル上のどの部分に変更を加えれば品質が向上するかを判別する難しさが残る．これに対しては，本研究の DSML にて，開発者が変更可能であり，品質に影響を与える要素を明示し，各要素がどの様な品質に，どの様な影響を与えるのかを分析した結果から，品質改善のためのパターンを提示することを考えている．例えば，どのようなルーティングプロトコルを選択するとどのような品質が改善／悪化するかを提示するパターンが挙げられる．このようなパターンを作成する際には，7.1.1 節の分析の項で述べた様な研究が有用であると考えられる．このパターンにより，目的の品質を向上させるためにモデル上のどの要素に変更を加えるか，という決定を支援することが可能になると考えている．

更には，このような設計の決定と変更を自動化することも将来課題として考えられる．すなわち，要求と設計変更との対応や，設計変更に対応する実装の修正を整備，明示することで，実行時の監視結果を受け，要求を満たすような設計変更を発見し，対象とする設計変更を反映するための実装変更を出力する．このような，実行環境への自動での適応のため，自己適応システムのためのソフトウェア工学 [20] における知見を取り入れることが考えられる．例えば，要求モデルと設計モデルとの対応付けを明示する手法 [5] や，設計と実装との対応付けをコンポーネント指向開発の技術を取り入れることで実現する手法 [37] などが存在する．これらの技術を WSN 分野に適用することで，WSN ソフトウェアのドラスティックな変更による自動品質改善を支援可能になると考えている．

謝辞

本論文の執筆にあたり，学部生であった頃より 6 年間，指導教員として様々なご指導を賜り，研究を進める環境，外部発表の機会などを与えてくださいました，早稲田大学大学院基幹理工学研究科 情報理工学専攻の深澤良彰教授に深謝を申し上げます．また，他大学の学生である私に対しましても，丁寧にご指導，ご助言いただき，ご自身の研究室に所属する学生との議論，研究の場を与えてくださいました，国立情報学研究所／東京大学の本位田真一教授にも深謝申し上げます．両先生方の多大なご指導，ご鞭撻により今日まで研究を進め，本論文をまとめることが出来ました．厚く御礼申し上げます．本論文をまとめるにあたりましては，センサネットワーク，ソフトウェア工学それぞれのご専門の観点から，大変貴重なご助言を賜りました早稲田大学の菅原俊治教授，鷲崎弘宣准教授にも深く感謝致します．

さらに，学部生の頃より，本研究及び論文の細部から研究者としての規範までに渡りご指導いただいた国立情報学研究所の鄭顕志助教にも深謝申し上げます．研究発表の場などにおきまして，本研究について非常に有用なご指摘，ご意見をくださいました，国立情報学研究所 石川冬樹准教授，田辺良則特任教授，坂本一憲助教，早稲田大学 高橋竜一助教にも感謝致します．

同期の大須賀隆彦氏，熊木健太郎氏，白井盛太郎氏，渡辺敦氏，井上聖久氏，大橋昭氏，志水理哉氏，高橋周平氏，中野由貴氏，福留康之氏，村上真一氏をはじめとする早稲田大学 深澤研究室，鷲崎研究室の皆様方や諸先輩方には，大学院生活を通し，ともに研究に励みながらも，様々なご助力をいただきましたことに感謝の意を表します．更には，他大学の学生でありながらも，ともに議論し，ともに学んできた同期の穐山空道氏，前澤悠太氏，石野克徳氏，伊藤雅博氏，金子裕司氏，全泰賢氏，ChavilaiSombat 氏，堀越永幸氏，藤原誠氏，吉池弘樹氏，宮前志津氏や，同じ研究グループとして，当時博士課程の学生の立場からもご指導いただきました鳥海晋博士，Valentina Baljak 博士をはじめとする東京大学 本位田研究室，電気通信大学 大須賀・田原研究室の皆様方，諸先輩方にも感謝致します．

最後に，長くに渡る学生生活及び研究生活を，心身両面に渡り支え続けてくれた家族に

感謝します.

参考文献

[1] Reza Abrishambaf, Majid Hashemipour, and Mert Bal. Structural modeling of industrial wireless sensor and actuator networks for reconfigurable mechatronic systems. *The International Journal of Advanced Manufacturing Technology*, Vol. 64, No. 5-8, pp. 793–811, 2013.

[2] Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pp. 1–6, New York, NY, USA, 2010. ACM.

[3] Bahar Akbal-Delibas, Pruet Boonma, and Junichi Suzuki. Extensible and precise modeling for wireless sensor networks. In Jianhua Yang, Athula Ginige, Heinrich C. Mayr, Ralf-D. Kutsche, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Information Systems: Modeling, Development, and Integration*, Vol. 20 of *Lecture Notes in Business Information Processing*, pp. 551–562. Springer Berlin Heidelberg, 2009.

[4] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, Vol. 40, No. 8, pp. 102 – 114, aug 2002.

[5] Germán H. Alférez and Vicente Pelechano. Dynamic evolution of context-a systems with models at runtime. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems*, Vol. 7590 of *Lecture Notes in Computer Science*, pp. 70–86. Springer Berlin Heidelberg, 2012.

[6] Danilo Ardagna, Elisabetta di Nitto, Parastoo Mohagheghi, Sébastien Mosser, Cyril Ballagny, Francesco D'Andria, Giuliano Casale, Peter Matthews, Cosmin-Septimiu Nechifor, Dana Petcu, Anke Gericke, and Craig Sheridan. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pp. 50–56, June 2012.

[7] Dhouha Ayed and Yolande Berbers. Uml profile for the design of a platform-independent context-aware applications. In *Proceedings of the 1st Workshop on MOdel Driven Development for Middleware (MODDM '06)*, MODDM '06, pp. 1–5, New York, NY, USA, 2006. ACM.

[8] Lan S. Bai, Robert P. Dick, and Peter A. Dinda. Archetype-based design: Sensor network programming for application experts, not just programming experts. In *Proceedings of the 2010 Fourth International Conference on Sensor Technologies and Applications*, IPSN '09, pp. 323–328, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Valentina Baljak and Shinichi Honiden. Discovery of configurations for indoor wireless sensor networks through use of simulation in virtual worlds. In *Proceedings of the 2010 Fourth International Conference on Sensor Technologies and Applications*, SENSORCOMM '10, pp. 323–328, Washington, DC, USA, 2010. IEEE Computer Society.

[10] Kai Beckmann and Marcus Thoss. A model-driven software development approach using omg dds for wireless sensor networks. In Sang Min, Robert Pettit, Peter Puschner, and Theo Ungerer, editors, *Software Technologies for Embedded and Ubiquitous Systems*, Vol. 6399 of *Lecture Notes in Computer Science*, pp. 95–106. Springer Berlin / Heidelberg, 2011.

[11] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, Vol. 10, No. 4, pp. 563–579, August 2005.

[12] Urs Bischoff and Gerd Kortuem. A state-based programming model and system for wireless sensor networks. In *Pervasive Computing and Communications Workshops,*

2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on, pp. 261–266, March 2007.

[13] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches — a survey. *Annals of Software Engineering*, Vol. 10, No. 1-4, pp. 177–205, 2000.

[14] Pruet Boonma and Junichi Suzuki. Bisnet: A biologically-inspired middleware architecture for self-managing wireless sensor networks. *Comput. Netw.*, Vol. 51, pp. 4599–4616, November 2007.

[15] Hugo Bruneliere, Jordi Cabot, Gregoire Dupe, and Frederic Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, Vol. 56, No. 8, pp. 1012 – 1032, 2014.

[16] Doina Bucur, Giovanni Iacca, Giovanni Squillero, and Alberto Tonda. The tradeoffs between data delivery ratio and energy costs in wireless sensor networks: A multi-objectiveevolutionary framework for protocol analysis. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pp. 1071–1078, New York, NY, USA, 2014. ACM.

[17] Matteo Ceriotti, Michele Corra, Leandro D'Orazio, Roberto Doriguzzi, Daniele Facchin, Stefan Guna, Gian Paolo Jesi, Renato Lo Cigno, Luca Mottola, Amy L. Murphy, Massimo Pescalli, Gian Pietro Picco, Denis Pregnolato, and Carloalberto Torghele. Is there light at the ends of the tunnel? wireless sensor networks for adaptive lighting in road tunnels. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pp. 187–198, April 2011.

[18] Matteo Ceriotti, Luca Mottola, Gian Pietro Picco, Amy L. Murphy, Stefan Guna, Michele Corra, Matteo Pozzi, Daniele Zonta, and Paolo Zanon. Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pp. 277–288, Washington, DC, USA, 2009. IEEE Computer Society.

[19] Dazhi Chen and Pramod K Varshney. Qos support in wireless sensor networks: A survey. In *International Conference on Wireless Networks*, Vol. 233, pp. 1–7, 2004.

[20] Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, HolgerM. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, HausiA. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, Vol. 5525 of *Lecture Notes in Computer Science*, pp. 1–26. Springer Berlin Heidelberg, 2009.

[21] Octav Chipara, Chenyang Lu, Thomas C. Bailey, and Gruia-Catalin Roman. Reliable clinical monitoring using wireless sensor networks: experiences in a step-down hospital unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pp. 155–168, New York, NY, USA, 2010. ACM.

[22] Paolo Costa, Luca Mottola, Amy L. Murphy, and Gian Pietro Picco. Programming wireless sensor networks with the teenylime middleware. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, Middleware '07, pp. 429–449, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[23] Priscilla Dantas, Taniro Rodrigues, Thais Batista, Flávia C. Delicato, Paulo F. Pires, Wei Li, and Albert Y. Zomaya. Lwissy: A domain specific language to model wireless sensor and actuators network systems. In *Software Engineering for Sensor Network Applications (SESENA), 2013 4th International Workshop on*, pp. 7–12, May 2013.

[24] Prem Devanbu, Sakke Karstu, Walcelio Melo, and William Thomas. Analytical and empirical evaluation of software reuse metrics. In *Software Engineering, 1996., Proceedings of the 18th International Conference on*, pp. 189–199, Mar 1996.

[25] Edsger W. Dijkstra. The structure of “the”-multiprogramming system. *Commun. ACM*, Vol. 26, No. 1, pp. 49–52, January 1983.

[26] Kishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Software Engineering for Sensor*

Network Applications (SESENA), 2012 Third International Workshop on, pp. 1–7, June 2012.

[27] David M. Doolin and Nicholas Sitar. Wireless sensors for wildfire monitoring. In *Proceedings of SPIE*, 2005.

[28] Adam Dunkels, Björn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 455–462, Nov 2004.

[29] Emadsamuelmlki Ebeid, Franco Fummi, Davide Quaglia, and Francesco Stefanni. Refinement of uml/marte models for the design of networked embedded systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pp. 1072–1077, March 2012.

[30] Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Jon Crowcroft. A shared sensor network infrastructure. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pp. 367–368, New York, NY, USA, 2010. ACM.

[31] Andy Evans, MiguelA. Fernandez, and Parastoo Mohagheghi. Experiences of developing a network modeling tool using the eclipse environment. In *Model Driven Architecture - Foundations and Applications*, Vol. 5562 of *Lecture Notes in Computer Science*, pp. 301–312. Springer Berlin Heidelberg, 2009.

[32] Moritz Eysholdt and Heiko Behrens. Xtext: Implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pp. 307–309, New York, NY, USA, 2010. ACM.

[33] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pp. 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[34] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, PLDI '03, pp. 1–11, New York, NY, USA, 2003. ACM.

[35] Nils Glombitza, Dennis Pfisterer, and Stefan Fischer. Using state machines for a model driven development of web service-based sensor network applications. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, SESENA '10, pp. 2–7, New York, NY, USA, 2010. ACM.

[36] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pp. 1–14, New York, NY, USA, 2009. ACM.

[37] Ning Gui, Vincenzo De Florio, Hong Sun, and Chris Blondia. Toward architecture-based context-aware deployment and adaptation. *Journal of Systems and Software*, Vol. 84, No. 2, pp. 185 – 197, 2011.

[38] Christian Hahn, Cristián Madrigal-Mora, and Klaus Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, Vol. 18, No. 2, pp. 239–266, 2009.

[39] Peter Harrop and Raghu Das. Wireless sensor networks (wsn) 2012–2022: forecasts, technologies, players – the new market for ubiquitous sensor networks (usn). Technical report, IDTechEx, Tech. Rep., Decermber 2012.

[40] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, pp. 28–41, New York, NY, USA, 2006. ACM.

[41] Soichiro Hidaka, Jean Bézivin, Zhenjiang Hu, and Frédéric Jouault. Principles and applications of model driven engineering (1) history and context of model driven engineering. *Computer Software*, Vol. 30, No. 3, pp. 3_25–3_44, 2013.

[42] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, Vol. 35, pp. 93–104, November 2000.

[43] Timothy W. Hnat, Tamim I. Sookoor, Pieter Hooimeijer, Westley Weimer, and Kamin Whitehouse. Macrolab: a vector-based macroprogramming framework for

cyber-physical systems. In *Proc. ACM Conf. on Embedded network sensor systems*, pp. 225–238, 2008.

[44] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pp. 471–480, New York, NY, USA, 2011. ACM.

[45] Michael Jackson. Aspects of abstraction in software development. *Software & Systems Modeling*, Vol. 11, No. 4, pp. 495–511, 2012.

[46] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pp. 254–263, New York, NY, USA, 2007. ACM.

[47] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.

[48] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. Dfuse: A framework for distributed data fusion. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pp. 114–125, New York, NY, USA, 2003. ACM.

[49] Nayanajith M. Laxaman, M. D. J. S. Goonatillake, and Kasun De Zoysa. Tikiridb: Shared wireless sensor network database for multi-user data access. *IITC*, 2010.

[50] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, Vol. 36, pp. 85–95, October 2002.

[51] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pp. 126–137, New York, NY, USA, 2003. ACM.

[52] Jun Bum Lim, Beakcheol Jang, Suyoung Yoon, Mihail L. Sichitiu, and Alexander G. Dean. Raptex: Rapid prototyping tool for embedded communication systems. *ACM Trans. Sen. Netw.*, Vol. 7, pp. 7:1–7:40, August 2010.

[53] Marcos López-Sanz, CésarJ. Acuña, CarlosE. Cuesta, and Esperanza Marcos. Uml profile for the platform independent modelling of service-oriented architectures. In Flavio Oquendo, editor, *Software Architecture*, Vol. 4758 of *Lecture Notes in Computer Science*, pp. 304–307. Springer Berlin Heidelberg, 2007.

[54] Fernando Losilla, Cristina Vicente-Chicote, Bárbara Álvarez, Andrés Ibora, and Pedro Sánchez. Wireless sensor network application development: An architecture-centric mde approach. In Flavio Oquendo, editor, *Software Architecture*, Vol. 4758 of *Lecture Notes in Computer Science*, pp. 179–194. Springer Berlin / Heidelberg, 2007.

[55] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, Vol. 30, pp. 122–173, March 2005.

[56] Luca Mainetti, Luigi Patrono, and Antonio Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, pp. 1–6, Sept 2011.

[57] Ivano Malavolta and Henry Muccini. A study on mde approaches for engineering wireless sensor networks. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pp. 149–157, Aug 2014.

[58] Quazi Mamun. A qualitative comparison of different logical topologies for wireless sensor networks. *Sensors*, Vol. 12, No. 11, pp. 14887–14913, 2012.

[59] Martin Manderscheid and Christian Prehofer. Network performance evaluation for distributed embedded systems using feature models. In *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*, pp. 46–55, 2013.

[60] Parastoo Mohagheghi and Vegard Dehlen. Where is the proof? - a review of experiences from applying mde in industry. In *Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, pp. 432–443, Berlin, Heidelberg, 2008. Springer-Verlag.

[61] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, MiguelA. F Nordmoen, and Mathias Fritzsche. Where does model-driven engineering help? experiences from three industrial cases. *Software & Systems Modeling*, Vol. 12, No. 3, pp. 619–639, 2013.

[62] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, Vol. 43, pp. 19:1–19:51, April 2011.

Lavagno, Laura Vanzago, and Stefano Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, pp. 515 –522, june 2008.

[64] Jishnu Mukerji and Joaquin Miller. Mda guide. version 1.0.1, 2003.

[65] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *Proc. Intl. Conf. on Information processing in sensor networks*, pp. 489–498, 2007.

[66] FelixJonathan Oppermann, CarloAlberto Boano, and Kay Römer. A decade of wireless sensing applications: Survey and taxonomy. In Habib M. Ammari, editor, *The Art of Wireless Sensor Networks*, Signals and Communication Technology, pp. 11–50. Springer Berlin Heidelberg, 2014.

l P. Midkiff. Efficient incremental code update for sensor networks. *ACM Trans. Sen. Netw.*, Vol. 7, pp. 30:1–30:32, February 2011.

[68] Nikolaos A. Pantazis, Stefanos A. Nikolidakis, and Dimitrios D. Vergados. Energy-efficient routing protocols in wireless sensor networks: A survey. *Communications Surveys Tutorials, IEEE*, Vol. 15, No. 2, pp. 551–591, Second 2013.

[69] Sung Park, Andreas Savvides, and Mani B. Srivastava. Sensorsim: a simulation framework for sensor networks. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, MSWIM '00, pp. 104–111, New York, NY, USA, 2000. ACM.

[70] Mallanagouda Patil and Rajashekhar C. Biradar. A survey on routing protocols in wireless sensor networks. In *Networks (ICON), 2012 18th IEEE International Conference on*, pp. 86–91, Dec 2012.

[71] A.R. Paulon, A.A. Frohlich, L.B. Becker, and F.P. Basso. Wireless sensor network uml profile to support model-driven development. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, pp. 227–232, July 2014.

[72] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Trans. on*, Vol. 55, No. 2, pp. 99–112, 2006.

[73] M. A. Razzaque, Chris Bleakley, and Simon Dobson. Compression in wireless sensor networks: A survey and comparative evaluation. *ACM Trans. Sen. Netw.*, Vol. 10, No. 1, pp. 5:1–5:44, December 2013.

[74] Taniro Rodrigues, Thais Batista, Flávia C. Delicato, Paulo F. Pires, and Albert Y. Zomaya. Model-driven approach for building efficient wireless sensor and actuator network applications. In *Software Engineering for Sensor Network Applications (SESENA), 2013 4th International Workshop on*, pp. 43–48, May 2013.

[75] Taniro Rodrigues, Priscilla Dantas, Flávia C. Delicato, Paulo F. Pires, Luci Pirmez, Thais Batista, Claudio Miceli, and Albert Zomaya. Model-driven development of wireless sensor network applications. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pp. 11–18, Oct 2011.

[76] Doug Rosenberg and Matt Stephens. *Use Case Driven Object Modeling with UMLTheory and Practice*. Apress, 2007.

[77] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pp. 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[78] Christopher M. Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pp. 265–278, New York, NY, USA, 2006. ACM.

[79] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, Vol. 40, No. 10, pp. 42–51, 2007.

[80] Swaytha Sasidharan, Fernando Pianegiani, and David Macii. A protocol performance comparison in modular wsns for data center server monitoring. In *Industrial Embedded Systems (SIES), 2010 International Symposium on*, pp. 213–216, July 2010.

[81] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, Vol. 39, No. 2, pp. 25–31, 2006.

[82] Winston Seah and Yen Kheng Tan, editors. *Sustainable Wireless Sensor Networks*, chapter 13. InTech, 2010.

[83] Leo Selavo, Gang Zhou, and John A. Stankovic. Seemote: In-situ visualization and logging device for wireless sensor networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pp. 1 –9, oct. 2006.

[84] Kewei Sha and Weisong Shi. Consistency-driven data quality management of networked sensor systems. *Journal of Parallel and Distributed Computing*, Vol. 68, No. 9, pp. 1207 – 1221, 2008.

[85] Akhilesh Shrestha and Liudong Xing. A performance comparison of different topologies for wireless sensor networks. In *Technologies for Homeland Security, 2007 IEEE Conference on*, pp. 280–285, May 2007.

[86] Doug Simon, Cristina Cifuentes, Dave Cleal, John Daniels, and Derek White. Java™ on the bare metal of wireless sensor devices: The squawk java virtual machine. In *Proceedings of the 2Nd International Conference on Virtual Execution Environments*, VEE '06, pp. 78–88, New York, NY, USA, 2006. ACM.

[87] Richard Soley. Model driven architecture. *OMG white paper*, Vol. 308, p. 308, 2000.

[88] Miroslaw Staron. Adopting model driven software development in industry: A case study at two companies. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *Model Driven Engineering Languages and Systems*, Vol.

4199 of *Lecture Notes in Computer Science*, pp. 57–72. Springer Berlin Heidelberg, 2006.

[89] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks: a survey. *ACM Trans. Sen. Netw.*, Vol. 4, pp. 8:1–8:29, April 2008.

[90] Harsh Sundani, Haoyue Li, Vijay K Devabhaktuni, Mansoor Alam, and Prabir Bhattacharya. Wireless sensor network simulators a survey and comparisons. *International Journal of Computer Networks*, Vol. 2, No. 5, pp. 249–265, 2011.

[91] Nguyen Xuan Thang, Michael Zapf, and Kurt Geihs. Model driven development for data-centric sensor network applications. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '11, pp. 194–197, New York, NY, USA, 2011. ACM.

[92] Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Heinzelman. Infrastructure tradeoffs for sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pp. 49–58, New York, NY, USA, 2002. ACM.

[93] Susumu Toriumi and Shinichi Honiden. Assignment of sensors for multiple tasks using path information. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pp. 120–127, Oct 2011.

[94] Martin Turon. Mote-view: a sensor network monitoring and management tool. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pp. 11–17, Washington, DC, USA, 2005. IEEE Computer Society.

[95] Naoyasu Ubayashi, Jun Nomura, and Tetsuo Tamai. Archface: A contract place where architectural design and code meet together. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 75–84, New York, NY, USA, 2010. ACM.

[96] Davya Upadhyay, P. Banerjee, and A.L.N. Rao. Critical performance comparison of on-demand routing protocols for optimal application in wireless sensor network. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference* -, pp. 462–466, Sept 2014.

[97] Hiroshi Wada, Pruet Boonma, Junichi Suzuki, and Katsuya Oba. Modeling and executing adaptive sensor network applications with the matilda uml virtual machine. In *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*, pp. 216–225, Anaheim, CA, USA, 2007. ACTA Press.

[98] Miao-Miao Wang, Jian-Nong Cao, Jing Li, and SajalK. Dasi. Middleware for wireless sensor networks: A survey. *Journal of Computer Science and Technology*, Vol. 23, No. 3, pp. 305–326, 2008.

[99] Qiang Wang, Yaoyao Zhu, and Liang Cheng. Reprogramming wireless sensor networks: challenges and approaches. *Netw., IEEE*, Vol. 20, No. 3, pp. 48–55, 2006.

[100] Tim Wark, Wen Hu, Peter Corke, Jonathan Hodge, Aila Keto, Ben Mackey, Glenn Foley, Pavan Sikka, and Michael Brunig. Springbrook: Challenges in developing a long-term, rainforest wireless sensor network. In *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, pp. 599–604, Dec 2008.

[101] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pp. 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[102] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pp. 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[103] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. Industrial adoption of model-driven engineering: Are the tools really the problem? In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems*, Vol. 8107 of *Lecture Notes in Computer Science*, pp. 1–17. Springer Berlin Heidelberg, 2013.

[104] Georg Wittenburg, Kirsten Terfloth, Freddy López Villafuerte, Tomasz Naumowicz, Hartmut Ritter, and Jochen Schiller. Fence monitoring: experimental evaluation of a use case for wireless sensor networks. In *Proceedings of the 4th European conference*

on Wireless sensor networks, EWSN'07, pp. 163–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[105] Fang-Jing Wu, Yu-Fen Kao, and Yu-Chee Tseng. From wireless sensor networks towards cyber physical systems. *Pervasive and Mobile Computing*, Vol. 7, No. 4, pp. 397 – 413, 2011.

[106] Xianghua Xu, Jian Wan, Wei Zhang, Chao Tong, and Changhua Wu. Pmsw: a passive monitoring system in wireless sensor networks. *Int. J. Netw. Manag.*, Vol. 21, pp. 300–325, August 2011.

[107] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, Vol. 52, No. 12, pp. 2292 – 2330, 2008.

[108] Yongjie Zheng and RichardN. Taylor. A classification and rationalization of model-based software development. *Software & Systems Modeling*, Vol. 12, No. 4, pp. 669–678, 2013.

業績一覧

論文誌

- **S-CLAIM: An Agent-based Programming Language for AmI, A Smart-Room Case Study**
  Valentina Baljak, Marius Tudor Benea, Amal El Fallah Seghrouchni, Cedric Herpson, Shinichi Honiden, Thi Thuy Nga Nguyen, Andrei Olaru, Ryo Shimizu, Kenji Tei, Susumu Toriumi
  Procedia Computer Science, Vol.10, 30–37, 2012.
- 自己適応ソフトウェアのための自己適応性設計に関する研究動向
  鄭 顕志, 清水 遼, 高橋 竜一, 石川 冬樹
  コンピュータソフトウェア，pp.49–59，Vol. 31 No. 1，2014 年 2 月.
- ○ 無線センサネットワークにおけるデータ品質改善の為の開発プロセス
  清水 遼, 鄭 顕志, 深澤 良彰, 本位田 真一
  電子情報通信学会論文誌 D vol.J97-D No.3, pp. 473–487, 2014.
- **Model-Driven-Development-based Stepwise Software Development Process for Wireless Sensor Networks**
  Kenji Tei, Ryo Shimizu, Yoshiaki Fukazawa, Shinichi Honiden
  IEEE Transactions on System, Man, and Cybernetics: Systems (TSMC), 2014.
  (DOI: http://dx.doi.org/10.1109/TSMC.2014.2360506)

査読付き国際会議

- ○ **Model Driven Development for Rapid Prototyping and Optimization of Wireless Sensor Network Applications**
  Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden
  The 2nd International Workshop on Software Engineering for Sensor Network

Applications (SESENA ’11), in conjunction with ICSE, Waikiki, Honolulu, Hawaii, pp. 31–26 May 21–28, 2011.

○ **Case Studies on the Development of Wireless Sensor Network Applications Using Multiple Abstraction Levels**
Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden
The 3nd International Workshop on Software Engineering for Sensor Network Applications (SESENA ’12), in conjunction with ICSE, Zurich, Switzerland, pp. 22–28, June 2–9, 2012.

○ **Data Quality-Centric Model-Driven Development Process for Wireless Sensor Network Applications**
Ryo Shimizu
The 12th ACM/IEEE Int. Conference on Information Processing in Sensor Network (IPSN 2013), Ph.D. Forum, Philadelphia, USA, April 7, 2013.

○ **Supporting Model Transformation Developments with Multi-Level Models: A Wireless Sensor Network Case**
Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden
The 11th International Conference Applied Computing 2014 (AC 2014), Porto, Portugal, pp. 29–36, October 25–27, 2014.

○ **Toward A Portability Framework with Multi-Level Models for Wireless Sensor Network Software**
Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden
The 2014 International Conference on Smart Computing (SMARTCOMP 2014), Hong Kong, China, pp. 253–260, November 3–5, 2014.

査読付きシンポジウム

○ 無線センサネットワークの為のモデル駆動開発に向けた **DSL** 非依存モデルの提案
清水遼, 鄭顕志, 深澤良彰, 本位田真一
マルチメディア, 分散, 協調とモバイル (DICOMO2010) シンポジウム, July 7-9, 2010.

査読無しシンポジウム

○ **Model-driven development for Rapid Prototyping and Optimizing Wireless Sensor Network Application**
Ryo Shimizu
Sixth joint NII-LIP6 Work Shop on Multi-Agent and Distributed Systems, Paris, France, October 10-11, 2011.

○ 最新の研究動向と **GRACE** センターでの研究成果 モデリング/プログラミング
清水遼
Workshop on New-Generation Networked Computing Paradigms: from Sensors to Web Services, 先端ソフトウェア工学に関する GRACE 国際シンポジウム 2010, Tokyo, Japan, 2010.

テクニカルレポート

○ **Meta-Models for Wireless Sensor Network Applications: Data, Group, and Node Views**
Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden
GRACE-TR 2012-01, GRACE Center, National Institute of Informatics, February 2012. 9 pages.