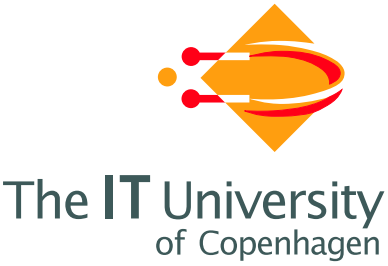


Progress Report

Bigraphical Location Models

Ebbe Elsborg

IT University Technical Report Series **TR-2006-94**

ISSN 1600–6100 **September 2006**

ISSN 1600–6100

ISBN 87-7949-138-3

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web: **www.itu.dk**

Preface

This progress report was submitted in partial fulfilment of the part A requirements of the Ph.D. Curriculum of the 4-year programme at the IT University of Copenhagen. It was prepared for my qualification exam, which is a prerequisite for entering part B of the programme. It describes the core of my research in the first two of four years of my Ph.D. studies and counts as my Master's thesis. A part of the work, namely chapter 3, was done in collaboration with other researchers as mentioned by that chapter's introduction. The Ph.D. Curriculum requires the report to detail directions for future work.

Bigraphical Location Models

Ebbe Elsborg

IT University of Copenhagen
Programming, Logic, and Semantics group
elsborg@itu.dk

6th September 2006

Contents

Abstract

In this progress report we begin evaluation of how well-suited Høgh Jensen and Milner's *bigraphical reactive systems* (BRSs) [JM04] are for modelling context-aware computing in ubiquitous systems. In this work we concentrate on the *location* aspect of context. First, we introduce the setting, motivate our work, and state our hypothesis. Then we present a digest of the research literature on *location models* forming a knowledge base for the rest of the report. We continue by developing bigraphical models of *context-awareness* and argue that these so-called *Plato-graphical* models constitute a proper foundation for modelling and simulating context-aware systems. A feature is that different calculi or programming languages can be combined in one model. Subsequently we define and analyse an encoding of a MiniML-like calculus with references in bigraphs (BRSs). This is needed for our implementation of a representative, minimalistic location model as a Plato-graphical model. Finally, we compare our approach to related work within *context calculi*, give directions for future work, and conclusions.

Chapter 1

Introduction and Motivation

1.1 The setting

This section aims to give an overall motivation for our work. Our work humbly tries to contribute to the foothill project "Analysing Movement in a Sentient Environment"[1], which is part of the UK Grand Challenges (GCs) of Computing Research[2].

Describing how our work fits into a larger perspective serves as an argument that our work is relevant. First, we use roughly three pages for presenting the setting before turning toward our more specialised field of research.

We first sketch the GC of interest and then the foothill project.

1.1.1 Global ubiquitous computing: Design and science

The GC in question is that of "Global Ubiquitous Computing: Design and Science" [CCK+05]. *Ubiquitous computing* (UC) is also called *pervasive computing*. UC was envisioned by Mark Weiser [Wei93, Wei91] to be the third wave of computing (after mainframes, and then personal computers), in which each person has many computers, receding into the background, at their disposal. An essential property of *ubiquitous computing systems* (UCSs) will be extreme dynamics, i.e. a rapidly changing network topology. The total aggregation of UCSs may be called the "Global Ubiquitous Computer". It is predicted that mobile devices will become more numerous, much smaller, and deeply embedded in many of the objects in everyday life, e.g. clothing or even in our bodies. We believe that this prediction will be realised. Thus, we need scalable design principles. Analysing UCSs will be difficult because UCSs will likely be distributed, mobile, and evolutionary. Furthermore, these devices can perform computation and communicate with each other, while being context-aware (location-aware). They may even be self-aware, thus exhibiting introspective behaviour. Imaginable is also for them to become self-organising and self-repairing.

[1] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-movement.html
[2] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/

Finally, trust, privacy, security, and reliability will become very important to guarantee seeing how these pervasive devices will be "everywhere" doing "everything". The science, toolkits, and theories for global computing do not yet exist. Progress has been made, but we shall need much more supportive science to really influence engineering of the technologies and devices ensuring sufficiently correct behaviour. We believe that without rigorous analysis of the possible interactions in a UCS behaviour will all too often be incorrect with fatal consequences. Theory and engineering should be a combined effort to realise the potential of *global ubiquitous computing* (GUC). More specifically, the aim is to define a set of design principles for GUC, and to develop science whose concepts, calculi, theories, and automated tools allow *predictive* analysis of GUC. These are the ideal goals of this GC.

1.1.2 Analysing movement in a sentient environment

The foothill project mentioned above is about *sentient computing* [ACH+01, Hop00] where it is proposed that software applications can be made more responsive and useful by observing and reacting to the physical world. Sensors (hardware) collect context (e.g. location) information of physical objects such as mobile devices and even other (mobile) sensors. We use the words *sentient* and *context-aware* interchangeably. It is fair to say that context-aware computing is a facet of UC. The foothill project aims to arrive at a conceptual framework in which to express a variety of rules of motion and interconnection, allowing context-aware systems to be programmed conveniently, simulated, and analysed rigorously. We use the term "system" broadly to mean a group of independent but interrelated elements comprising a unified whole. We will detail this in chapter 2. The framework could consist of a calculus and a derived programming language along with a programming methodology so that the language may be used and evaluated by people whose primary interest is in applications. The ultimate goal of this foothill project is to unify theory and practice in sentient computing. A step toward this goal could be to model and program a sentient or "reflective" building, where sensors continually transmit data to a monitor that maintains a data structure of the locations of physical objects. A more advanced task could be to also model mobile virtual objects such as mobile code moving from one software domain to another. We envision *bigraphical reactive systems* (BRSs) due to Milner and Høgh Jensen [JM04], or some extension of this theory, as a suitable framework for context-aware computing.

1.1.3 Bridging theory and practice

Recently, [Ter06] argued that combining theory and systems building (wrt. trust) is important because we need to establish a basic understanding and an appropriate level of abstraction. It is essential for systems builders and theoreticians to collaborate successfully, and this requires consensus on definitions of core concepts in the problem domain.

The importance of combining engineering with foundational work in realising

computing in space is also emphasised in [Mil02]. "Computing in space" is about communication across space and actually considers the global computer as both a physical and a virtual entity. The term *infodynamics* is used to describe the fact that physical devices move in physical space but also in virtual space via their representations. *Infostatics* is the term used to state that software superposes virtual space upon physical space. The conclusion is that joining the forces of software engineering and software theory is necessary to achieve success with the global computer.

When programming context-aware systems the programmer needs commonly-recognised abstractions and design patterns [Leo98]. Formal methods aid in understanding the essence of the programming task [JPR04].

Having given a very high-level motivation we next review the challenges for theory in a little more detail. The purpose is to make the issues discussed so far more concrete. There are also challenges for engineering, but in this introduction we focus on the theoretical part, and refer the reader to [CCK+05] for an outline of the challenges for engineering.

1.1.4 Theory

As mentioned we need a conceptual framework enabling rigorous analysis supported by techniques and automated tools. Many topics can be discussed in this regard; structure, information flow, mathematical analysis, and methodologies and tools. We briefly sketch these.

Structure refers to the ways entities (e.g. devices) interact and move among each other, as captured by structural theories of processes. Both physical and virtual space is relevant, and three issues are important, namely placing, linking, and mobility. [CCK+05] envisions that no later than the year 2010 will we have a calculus or logic which allows for experimentation with modelling prototypical systems such as a sentient building, or even more complex scenarios. We would like to take steps towards this goal, and have begun to do so in the work presented in this report. We expect that models for real-life systems will need to capture not only *time* as a continuous variable, but also *continuous space*; according to [CCK+05] one approach is to use *hybrid automata* (modelling/representing both time and continuous space) governed by differential equations. Also, *stochastics* (propabilistic) will probably be needed to, e.g. faithfully model and simulate device movement in a sentient building, because sensors are not perfect.

Regarding information flow, we find that the need to query distributed data is paramount. As mentioned in [CCK+05], one can expect a merging of research on semi-structured data and process models to handle that movement of data and processes is becoming alike. An example of such work is Reactive XML [HNO06, HNOW05], which is a bigraph-and-XML-based approach. Recently, modelling and verification of protocols for communication in mobile ad hoc networks (MANETs) has become a lively research area, where issues of trust and resource access challenge formal models.

Mathematical analysis of complex systems such as UCSs should be driven by

experimental research because it seems impossible to foresee all the potential and problems of this new computing paradigm. Real systems should be modelled and analysed, and hopefully theory can impact the way engineers build systems. A help in understanding UCSs could be a graphical representation and reconfiguration of, e.g. network topology. We would like to contribute here also by gradually formalising and reasoning about increasingly realistic systems; a first step is taken in this report. One could also imagine a family of models with consistency requirements between them, where each model aids in reasoning about a certain level of a UCS.

This concludes the overall motivation. A motivation more specific to our work will be given in the chapters 2 and 3. In the the following sections we narrow the problem domain, state our hypothesis, and describe how we approach the task of testing the hypothesis.

1.2 Narrowing the problem domain

GUC is a vision rather than a concrete research problem that we can solve or answer. To obtain a more tangible problem to attack we narrow the domain of investigation. An important facet of UC that has received much attention in the research literature is context-aware computing, where entities (e.g. mobile devices) are aware of their surroundings, i.e. adapt their behaviour depending on the context at hand [SAW94], interpreting "context" to mean the situation in which the computation takes place [DA00]. Context-aware systems typically have a component that maintains a model of the current context, and such components are known as *context models* [HIR02]. The most commonly exploited instance of context is physical location, as witnessed by the literature and the context-aware systems and toolkits that have been implemented. Location-aware applications acquire information from sensors, which can happen in a uniform way through a *location model* [BD05] that interprets sensor information to maintain a model of the current locations (positions) of, e.g., mobile devices. We delve into location models in chapter 2. Context models and location models are concrete enough for us to study and they are an essential part of context-aware systems.

1.3 The hypothesis

Our point of origin is the theory of bigraphs [JM04, Mil05a, JM03]. A principal aim of BRSs is to model ubiquitous systems. In this report we begin evaluation of this aim. Thus, the main hypothesis of our work is:

Hypothesis (main). *BRSs are suitable for programming, simulating, and reasoning rigorously about ubiquitous systems.*

In this report we begin to test this hypothesis by ascertaining a more tangible hypothesis that supports the main hypothesis:

Hypothesis (supporting). *BRSs are suitable for modelling location-aware systems.*

To ascertain this supporting hypothesis we endeavour modelling location-aware system directly in bigraphs. (We overload the term "bigraphs" to mean the theory of bigraphs including BRSs.) In the chapters 3 through 5 we find that:

1. It is awkward to model context-aware (location-aware) systems directly in bigraphs.
2. This awkwardness can be alleviated by using *Plato-graphical* models.
3. Location models can be modelled as Plato-graphical models using a bigraphical encoding of a MiniML-like calculus with references.

We argue that these findings ascertain our supporting hypothesis. To strengthen the supporting hypothesis we propose to study the following two questions:

- Are Plato-graphical models useful for simulation of context-aware systems?
- Is there a need for a notion of bisimilarity between BRSs to support formal reasoning about context-aware systems in Plato-graphical models?

We touch upon these two questions in this report, but leave their treatment for future work.

1.4 Our approach

This section outlines our strategy for testing the supporting hypothesis stated above. We take an *experimental* approach by comprehensive modelling of a concrete realistic system. Our strategy for testing the supporting hypothesis is detailed as follows.

First, we study the literature on location models to gain understanding of the models we wish to model bigraphically. (1) To test whether bigraphs are suited for direct modelling of context-aware systems, we pick some queries on location-aware systems to model. Such queries are concrete, and it is clear when a query has been implemented faithfully. (2) Finding that this direct modelling is inconvenient motivates a more advanced modelling method; Plato-graphical models. We model a context-aware printing example from the literature to test the Plato-graphical models. (3) Further, we use our gained knowledge of location models to formulate a representative one, and model it as a Plato-graphical model. In doing this, we define an encoding of a MiniML-like calculus into bigraphs, and analyse this encoding.

In short: In this report we essentially invent and challenge Plato-graphical models (and thus BRSs) by testing how suited they are for modelling location models.

1.5 Outline of this report

Having introduced and motivated our domain of work we proceed as follows. In chapter 2 location models are investigated. Having this piece of background knowledge in place we proceed to define Plato-graphical models for context-aware systems in chapter 3. These models serve as basis for the chapters 4 and 5. In chapter 4 we encode a calculus with references in bigraphs because this is needed for the bigraphical location model presented in chapter 5. Chapter 6 presents related work in some detail, and chapter 7 extensively discusses directions for future work. Finally, we conclude in chapter 8.

We stress the fact that this is a *progress report* so some of the work presented here is *in progress*. We remark that the chapter on related work takes up significant space because becoming knowledgeable within the relevant research areas has been a priority in this study, and should serve as a basis (and catalyst) for our further research.

Chapter 2

Location Models

2.1 Introduction

It is well agreed upon that *location* is an important context-parameter [Sch95, Leo98], but not the only one [SBG99] in context-aware computing, and that context-aware computing will become increasingly important in the years to come. This chapter serves as general background knowledge for chapter 3 on bigraphical models of context-aware systems, and as basis for modelling a location model in chapter 5.

2.1.1 Location systems

First, we need a piece of terminology.

2.1.1.1 Located-objects

As mentioned earlier, there are (at least) two ways to think about location; namely physical and virtual. In the present discussion we think of physical location, i.e. the location of objects in the physical world. Typically, it is the location of real-world entities such as mobile devices (e.g. mobile phones) that is interesting for location-aware applications (which we explain shortly). Following [Leo98, ST94] we use the term *located-object* to refer to a mobile object whose physical location can be tracked.

2.1.1.2 The overall location system model

In this chapter we present a digest of the research literature on *location models*. We say that a location model is constituted by (1) representations of static and mobile real-world objects, (2) spatial relationships between these objects, (3) a collection of rules that model object movement, and (4) a collection of location information queries on the model. Location models are essential parts of *location systems* (see

[HB01] for a survey on location systems) because they provide a uniform way for applications to obtain location information, which facilitates rapid development. Before delving into location systems we need to address how information about location is presented in different formats. *Geometric* coordinates, as used by the Global Positioning System (GPS), refer to a point or geometric figure in a multi-dimensional space. *Symbolic* coordinates are names and can refer to cell-IDs in cellular networks such as the Global System for Mobile communications (GSM) or Wireless Local Area Networks (WLANs), or to radio frequency tags (RFIDs). The distinction between these two coordinate types is fundamental and we will return to it shortly. For now, please consider figure 2.1, which depicts the overall system model. We explain

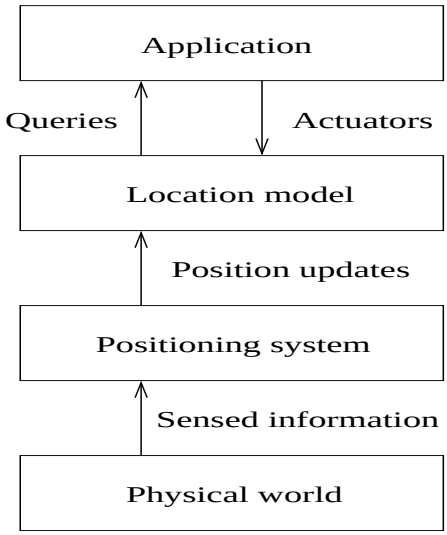


Figure 2.1: Overall location system model.

figure 2.1 in a top-down fashion. A *location-aware application* (see [Leo98] for examples) queries a location model for location information. By location-aware we mean "the ability to adapt behaviour to the physical locations of users, resources, and processes" [Leo98]. In some cases the application can update the location model; we say that this is *actuation*. The different kinds of queries and actuators imply demands on the internal structure and organisation of the location model. The location model maintains a representation of the state of the physical world by receiving events about updated position information on mobile objects from a *positioning system* (see [HB01] for an overview of positioning systems). "A positioning system allows a mobile object or tracking system to issue a position update with a coordinate identifying a location to the location model." [BD05]. [Leo98] states that a positioning system measures the location of the querying located-object (e.g. vehicle navigation systems), whereas a tracking system measures the location of *other* located-objects

(e.g. the Active Badge system [WHFG92]). We do not wish to distinguish between tracking and positioning because we need objects to enquire about both their own and other objects' locations. If different positioning systems are in play then there is a need for *sensor fusion*, but that is out of the scope of this report so we refer to [HBB02]. In the preceding explanation we have used the terms "location" and "position". Following [HB01] we distinguish between *physical position* and *symbolic location*; a physical position is specified by a geometric coordinate, whereas a symbolic location is specified by a symbolic coordinate. The positioning system generates location information events on the basis of what its (hardware) *sensors* sense in the physical world. The sensors track the movement of located-objects and the sensed information is delivered to the location model. The physical world is the world we live in, which is narrowed according to the geographical location of interest, e.g. a sentient building.

We sometimes wish to speak of a geographical point or area without being specific as to whether we consider it from a geometric or symbolic point of view. We overload the term "location" for this purpose.

2.1.1.3 Focus

We focus on a conceptual classification of the models. We do not discuss specific location-aware applications, positioning systems or sensor technology any further, except for a few comments later on.

2.2 Relationships, queries, and requirements

As mentioned earlier, location-aware applications query a location model. We intend to identify types of common queries, and mention which demands they list for the underlying location model. This requires us to first study coordinates and spatial relationships between locations. We do not consider actuation in this chapter, but we do return to it briefly during this report and in chapter 7.

We proceed to explain location models and their basis.

2.2.1 Basic properties of coordinates

We follow the definitions of [BD05]. A coordinate is an identifier specifying the physical position of an object wrt. a given *coordinate system*, or the symbolic location by a *name* (e.g. a cell-ID). A coordinate system is a set of coordinates. There are essentially two different classes of coordinates, namely *geometric* and *symbolic*. We discuss each class of coordinates in turn.

2.2.1.1 Geometric coordinates

Geometric coordinates refer to a point or geometric figure in a multi-dimensional space and can be *global* or *local*. Geometric coordinates naturally support *calcula-*

tion of physical distance and containment relationships between positions (which are described by one ore more coordinates), to which we return shortly. Through calculation, geometric coordinates also support the following operations: Area overlap, areas touching, and area containment.

GPS is an example of a system using global geometric coordinates, where coordinates are triples of longitude, latitude, and elevation above main sea level. Many applications use GPS – e.g. navigation systems in cars. An example of a system using local geometric coordinates is the Active Bat system [ACH+01], which is a high-resolution indoor positioning system providing three-dimensional coordinates wrt. a local Cartesian coordinate system. In other words the physical space is defined by a coordinate system, positions are identified by coordinate tuples, and the location model is geometric, i.e. identifies pisitions by geometric coordinates.

2.2.1.2 Symbolic coordinates

Symbolic coordinates are *names* that refer to locations, e.g. a room, a cell ID, or an IR identifier of a sensor. A reason for having symbolic coordinates is that they are "human-readable" – it is often more useful to know that a person is in a particular cell (e.g. a room), than at some given (set of) coordinate(s). Given only symbolic coordinates, it is *not* possible to *calculate* distances via a distance function, but the distance and containment relationships on locations (to be explained shortly) must be represented explicitly in the location model. Furthermore, a symbolic notion of *nearness* (or *proximity*) can be supported, i.e. located-object is close to another located-object or location. We return to this below.

The Active Badge system [WHFG92] provides symbolic identifiers (coordinates) for locations via fixed IR sensors registering users' badges that transmit a unique identifier. In other words the location space is defined by the placement of fixed sensors, a location is defined by the symbolic name of the sensor, and the location model is symbolic. Another application that uses symbolic coordinates is Active Office system [WJH97] where locations are denoted "building", "floor", "room" and so forth.

2.2.2 Relationships wrt. locations and located-objects

In the literature five relationships pertaining to locations and located-objects are emphasised as having practical importance; *Contains* (inclusion), *connected-to*, *near* (proximity), *range*, and *distance*. These spatial relationships between locations are relevant for queries and topologies of location models. We briefly discuss each one in turn.

Contains Indicates whether a location is completely included in another. This relation is supported naturally in models with a hierarchical location structure such as trees and lattices. As mentioned, it can be calculated in geometric models. As an example: A building can contain a room, but hardly vice versa.

Connected-to Refers to some linking between locations. This relationship is often captured by introducing a graph-based location structure, as it can not be calculated or derived. Examples: Two mobile devices can be connected, e.g. via Bluetooth. Two rooms can be connected by a door.

Distance The distance relationship is defined on spatial objects and is usually expressed as a natural or real number. It can be calculated in geometric models, but needs to be explicit in symbolic models. As an example: Two mobile devices can be positioned ten meters from each other, but in different rooms. This raises the question of how to calculate distance; by following a path via the connected-to relationship, or as the Eucledian distance. We return to this question in chapter 5.

Near For a located-object l to be near another, a notion of "distance function" is required. The near relationship could contain the n located-objects closest to the position of l. This relationship can be calculated in geometric models, but must be explicitly represented in symbolic models. It can be seen as a specialisation of the distance relationship. As an example: A user of a mobile device may want to find the nearest printer.

Range The range relationship has the located-objects within a certain geographic area of the located-object in question. To support this query located-object positions must be known and the contains relationship modelled, i.e. it has to be defined whether a coordinate lies within a spatial area. As an example: The sending of messages to receivers in a certain geographic area, e.g. a room on (contained in) the fourth floor (contained) in a building.

2.2.3 Queries

In [BD05] four different query types, which location models should support, are identified. We present and explain them for future reference. When explaining the queries we refer to the relationships.

Position queries: Determination of the position or location of a located-object like a user's mobile device, or a static object like a room. A position is defined by local or global coordinates. It would, e.g., be relevant with a local coordinate system for a moving train so that a traveller can be located in a compartment instead of his or her position on the ground [BD05]. To compare positions from local coordinate systems, mappings to a common global coordinate system must be defined.

Nearest neighbour queries: A search for the located-object or location closest to a certain position, e.g. a printer. Besides known located-object positions there is need for a *distance function* to support this query. This function should output the physical distance when supplied with two coordinate tuples.

Navigation: Finding paths between locations. There is a need to model the topological connected-to relationship, which describes interconnections between neighbouring locations. This can, e.g., be used to find the shortest or fastest path, or a path for a person in a wheel chair. One could imagine adding weights to links for this purpose.

Range queries: Search for all located-objects within a certain geographic area. This can, e.g., be used to send messages to receivers in a certain geographic area as in Geocast protocols [DR03]. To this end, the model needs to be able to determine the positions of the located-objects, and also the topological contains relationship, i.e. whether a coordinate lies within a spatial area. Containment is supported implicitly for geometric coordinates, but must be specified explicitly for symbolic coordinates.

These are the query types we will consider supporting in our bigraphical location model in chapter 5. We should mention that [BD05] also has a requirement stating that all information of the location model can be visualised, but we do not consider that as a query as such.

2.2.4 Location model requirements

Having described these queries, the following model requirements, not all of which need to be fulfilled at the same time, are derived in [BD05]. The requirements are on general-purpose models that wish to support all four query types, and so a model for a specific purpose need not necessarily fulfil these requirements to be of use. We believe that our preceding treatment justifies these requirements without further comments.

Object positions: Need geometric and symbolic coordinates to support a wide range of applications that have been implemented. Can do with either geometric or symbolic in some cases. Multiple local and global coordinate reference systems are desirable. This supports the position queries.

Distance function: Distances between spatial objects; Eucledian and desirably over paths. This is required for the nearest neighbour and geometric range queries. We argue that range can also be supported by location containment and can thus make sense in symbolic models also.

Topological relations: Contains and connected-to. These are needed to support range queries and navigation queries, respectively.

Orientation: Horizontal and vertical orientation is required for some applications, e.g. to determine which situation a person is in.

According to [Leo98] *co-location* is another interesting relative relationship. We consider this to be a range query where the range is exactly one's own location.

[BD05] argues that *minimal modelling effort* should be considered when constructing a location model, i.e. wrt. *accuracy* (creation and updating of the model, dynamics), *level of detail* (granularity of locations), and *scope* (the area covered; a building, a room, a country). We agree, and return to this in chapter 5.

2.3 Classification of location models

When classifying location models we need to have a clear terminology. Unfortunately, there is no clear consensus in the research literature regarding the terminology of location model types. We proceed by synthesising the terminology of the literature. Some terms used are: Geometric, physical, symbolic, geographical, semantic, metric, topological, and Cartesian [HB01, Rot03, Pra00, BS01, BD05, DRD+00, BZD02, CK00]. We believe that these terms, in essence, cover two different types of location models; symbolic and geometric. We group the terms as follows:

Symbolic includes geographical, semantic and topological.

Geometric includes physical, metric, and Cartesian.

Hybrid models are combinations of symbolic and geometric models.

We continue by giving explanations of geometric and symbolic location models following [Leo98], along with brief justifications of our grouping. We begin with the geometric models and continue with the symbolic models.

2.3.1 Geometric location models

Geometric location models define the physical space by one or more multidimensional reference coordinate systems. Both positions and located-objects are represented as points, areas, or volumes within these coordinate systems. This supports calculation of the relationships distance (which may not be accurate) and containment between positions, and therefore also allows for calculation of area overlap, and whether areas touch. The connected-to relationship is however not inherent. A geometric location model is said to be *unified* if it has multiple coordinate systems, otherwise *simple*. Often, *uncertainty areas* are used to capture the imprecision of sensors (see e.g. [SBG99, HHS+02, HB01]) when positioning located-objects. Some geometric systems use *global* coordinates, e.g. referring to the position on the Earth's surface like in GPS, while other systems (e.g. Active Bat [ACH+01, HHS+02]) use *local* coordinates referring to a (smaller) Cartesian coordinate system with another point of origin (typical for indoor positioning systems). Geometric models use *absolute* positions, i.e. located-objects and locations are positioned with reference to some common point of origin (within each coordinate system), and *not* relative to each other.

2.3.1.1 Physical, metric, and Cartesian models

Physical, metric, and Cartesian denote exactly the same type of model as geometric.

2.3.2 Symbolic location models

In symbolic location models locations and located-objects are referred to by symbols or *names* such as "Room 4C.16" or "Linus Torvald's laptop". Symbolic models use *relative* locations meaning that each located-object has its own frame of reference because there are no underlying absolute positions. Locations can be organised in different structures to support different queries. We review three different approaches: Set-based models, graph-based models, and hybrid models combining the two approaches.

2.3.2.1 Set-based models

Locations can be modelled (naturally) as sets of located-objects which are represented by symbolic coordinates. A located-object is a member of a location whenever it is physically within the associated area or volume. Using sets, overlap of locations L_1 and L_2 is represented by set intersection $L_1 \cap L_2 \neq \emptyset$, and thus also the containment relationship, if $L_1 \cap L_2 = L_1$ then L_2 contains L_1. This supports range queries by subset construction. It should also be possible to test for equivalence on locations. The support for queries related to spatial distances is naturally limited, but a notion of qualitative distance on symbolic coordinates can be modelled via set membership tests, we refer to [BD05] for the details. We mention two example systems; Guide [CDMF00] and Active Badge [WHFG92].

Cell models This is the most basic and flexible set-based model, and thus named a *simple symbolic* model. In this model the location space is described by cells so cells are the symbolic locations. A cell is a well-defined geographical area, e.g. a room. Cells can overlap, and need not cover the whole space. This is realistic wrt. sensor systems. There is typically no containment relationship in cell models. An example system is GPS, where the cell's area is a circle defined by the sighting coordinates and the accuracy margins [Leo98].

Zone models A zone model is a cell model with exclusive membership, i.e. non-overlapping locations. In a cell model, cells may overlap. These overlaps are named zones. Each zone is part of one or more cells. Now, zones are used as symbolic locations. Imposing the constraint that locations must be non-overlapping yields an *exclusive symbolic* model. A single zone space can accommodate an arbitrary number of cells, which is useful if several sensor systems are in play. Since zones do not overlap, a located-object can be in at most one zone at a time. As noted in [Leo98] the movements of one located-object can be modelled by a single finite-state machine making the zone space a natural framework for persistent tracking and

movement prediction. Imposing hierarchical locations via a partial order on a zone model yields a location tree structure. An example is found in [HHS+02] where a *quadtree* (a tree where all nodes have four children) is used. (Such structures support multi-resolution and thus scalability of design.)

Domain models A domain model is a zone model where locations (zones) have been partially ordered in a virtual hierarchy of domains. The aim is to enable multi-resolution tracking. A zone is a member of at most one domain. Domains are partially ordered, by the contains relationship, and can overlap. Thus, it is now possible to relate some zones to "building B", which is part of "Campus S" and also part of "The computer science department", for example. If "A" is a member of "B" then it is also a member of the ancestors of "B" in the domain ordering. Changes in domain membership should propagate through the model. Multi-resolution refers to the ability to, e.g., say that a located-object is situated in room "4C.16" or in "ITU" (where "4C.16" is a member of "ITU"). See [DR03] for an example. If a lattice is imposed as the ordering, then a simple notion of distance can be expressed: Given three locations l_1, l_2, l_3 we have that $distance(l_1, l_2) < distance(l_1, l_3)$ if $sup(\{l_1, l_2\}) < sup(\{l_2, l_3\})$ in the lattice. This may not be a very good metric, but hierarchical models do not provide means to model interconnections between locations.

2.3.2.2 Graph-based models

In the graph-based approach symbolic coordinates define the vertices V of graph $G = (V, E)$. An edge $e \in E$ is added between to vertices if a direct connection between those two vertices exist in the physical world. An edge could be a door between two rooms (vertices). Edges can be weighted (and oriented) to model distances. It is clear that this setup supports the connected-to and distance relationships explicitly. It is therefore well-suited for navigation and distance queries. The containment relationship is not supported, but can be simulated by linking from a reference vertex to all other vertices which are considered to be within a particular range. This is not a general mechanism though. Furthermore, locations can not consist of other locations. Examples can be found within "smart environments" [RLU94, OJDA01].

2.3.2.3 Combined graph- and set-based symbolic models

As seen, the set-based models support range queries well whereas the graph-based models support distance and connected-to. We wish to combine the two model types to obtain all the benefits. The set-based part of this hybrid model is a set of symbolic coordinates. Locations are sets of coordinates. Locations are connected by edges if a connection between these locations exists in the physical world. For instance, two rooms can be connected by a door, and two floors by a stairway. Edges can be weighted to model distances. We can introduce more than one graph to represent

views, i.e. different aspects of the world. An example is the Active Map system [Sch95].

2.3.2.4 Geographical, semantic, and topological models

Geographical models typically organise the location space *hierarchically* via identifiers such as "City of Copenhagen", "IT University of Copenhagen", and "The C wing".We consider these models to be a special case of symbolic models where the symbols happen to carry geographical meaning, and the location space is, e.g., organised as a tree.

We believe the term *semantic* location model was coined in [Pra00], where places (semantic locations) are represented by URIs and can have attributes indicating the nature or purpose, or even physical or geographical information. In [Pra00] there are three types of location; physical (grid based), geographical (hierarchical), and semantic (web like). Like in [Rot03], we do not distinguish semantic and geographical location. We consider semantic locations to be a special case of symbolic locations because web-like structures can well be described using graphs to model location space in a symbolic model. *Topological* [BS01] models are related to semantic models. Organising semantic locations in a hierarchy supports the containment relationship. Still, the combined model remains a special case of symbolic models.

2.3.3 Hybrid location models

Hybrid location models are so called because they are combinations of the two model types we have outlined above, namely symbolic and geometric. Symbolic and geometric models are orthogonal and can therefore be combined in numerous ways. Hybrid models aim to possess the advantages of both types of models, i.e. basically to provide applications with a high-level structured symbolic representation of locations while preserving the accuracy of location information inherent in geometric coordinates. This combination supports the queries we considered in section 2.2. The trade-off is a higher modelling effort.

In [BD05] it is suggested to add geometric information to a symbolic model. This can be done either for each symbolic location, or for only some of them. It is also possible to deduce symbolic locations from geometric locations, and relative from absolute by the containment relation [HHS+02]. It is a matter of abstracting certain geometric data into meaningful symbolic notions. Typically, a non-hybrid model will be geometric and absolute, or symbolic and relative.

One example of a hybrid model is found in [JS02], where in a symbolic tree model each node (location) has geometric information as an attribute, and queries such as distance and containment are supported. Another example is found in [Rot03], where a domain model is presented. This hybrid model has mappings between local and global coordinates, and also between geometric and symbolic locations. Local coordinates can be translated into global coordinates. Global geometric coordinates can then be translated into global symbolic coordinates, which in turn can be trans-

lated into global geometric *areas*. Such mappings are widely adopted in practise according to [BD05].

2.3.4 Views

An idea found in several papers [BD05, BBR02, BS01, Rot03] is that of having *views*, i.e. having multiple hierarchies e.g. representing different views of an organisation; the building, employee relationships, access control et cetera. This thought also appears in theoretical work, which we treat in chapter 6.

2.3.5 Location-aware systems

Several *location-aware systems* (or *location systems*) have been implemented demonstrating the feasibility of using location information in practise while challenging existing and developing new technology. By location system we mean a computer system that via hardware sensors can track the physical location of objects (to be explained shortly), and is able to output this information in some suitable format. See [HB01] for a survey of location systems where they have been categorised according to their properties geometric/symbolic and absolute/relative. Location systems as such are not within the focus of this report so we refrain from further discussion.

2.4 A model of a reflective building

Describe a reflective building as "one equipped with sensors, which continually transmit data of the building's occupancy to a monitor that maintains a data structure which faithfully records the occupancy."[1]. This is a very loose description. We come a little closer to more tangible properties in [Hop00], where a sentient building is said to support containment, proximity (near), and coordinate systems. Except for the coordinate systems it does seem that we can do with a very simple location model, e.g. a symbolic tree model. This is the starting point for this report. We do, however, wish to model more realistic (complex) examples in our work so coordinates should be considered. We will come back to this in chapter 5.

2.5 Concluding remarks

We hope to have given the reader sufficient background knowledge for the next chapters. The most important points are:

- Location models facilitate efficient development of location-aware applications.

[1] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-movement.html

- To support a broad range of applications a location model should support four query types (position, near, navigation, range), and thus be a hybrid model (geometric and symbolic). Queries rely on certain spatial relationships.
- We have briefly noted some thoughts about a location model for a reflective building – namely that a symbolic tree model is a good starting point for a reflective building.

We have tried to cover the literature on location models. Much can and has been written about different implementations of location-aware applications. To limit ourselves we have decided to leave those out of the focus in this report. The same goes for positioning systems and sensor technologies underlying a location model. The focus is on modelling of location models.

Chapter 3

Bigraphical Models of Context-Aware Systems

This chapter consists of a conference paper [BDE+06] followed by the appendices of a technical report [BDE+05]. The only changes made are minor typographical ones, and adjustment of references to fit in this report. Both the conference paper and the technical report were developed and produced in cooperation with Lars Birkedal, Søren Debois, Thomas Hildebrandt, and Henning Niss. All authors contributed equally.

The work in this chapter is foundational for chapter 5 where a bigraphical location model is presented, in that it develops the modelling technique used, namely bigraphical models of context-aware systems. Such models can be used to model location models. It also supports chapter 4 with a gentle introduction to bigraphs.

Abstract

As part of ongoing work on evaluating Milner's bigraphical reactive systems, we investigate bigraphical models of *context-aware systems*, a facet of ubiquitous computing. We find that naively encoding such systems in bigraphs is somewhat awkward; and we propose a more sophisticated modelling technique, introducing *Plato-graphical models*, alleviating this awkwardness. We argue that such models are useful for simulation and point out that for reasoning about such bigraphical models, the bisimilarity inherent in bigraphical reactive systems is not enough in itself; an equivalence between the bigraphical reactive systems themselves is also needed.

3.1 Introduction

The theory of *bigraphical reactive systems*, due to Milner and co-workers, is based on a graphical model of mobile computation that emphasizes both locality and connectivity [JM04, Mil04c, Mil05c]. A bigraph comprises a place graph, representing locations of computational nodes, and a link graph, representing interconnection of these nodes. We give dynamics to bigraphs by defining reaction rules that rewrite bigraphs to bigraphs; roughly, a bigraphical reactive system (BRS) is a set of such rules. Based on methods of the seminal [LM00], any BRS has a labelled transition system, the behavioural equivalence (bisimilarity) of which is a congruence.

There are two principal aims for the theory of bigraphical reactive systems: (1) to model ubiquitous systems [Wei93], capturing mobile locality in the place graph and mobile connectivity in the link graph; and (2) to be a meta-theory encompassing existing calculi for concurrency and mobility. To date, the theory has been evaluated only wrt. the second aim: We have bigraphical understanding of Petri nets [Mil04b], π-calculus [Jen06, JM04, JM03], CCS [Mil05c], mobile ambients [Jen06], HOMER [BH06], and λ-calculus [Mil04c, Mil05b].

The present paper initiates the evaluation of the first aim. We investigate modelling of *context-aware systems*, a vital aspect of ubiquitous systems. A context-aware application is an application that adapts its behaviour depending on the context at hand [SAW94], interpreting "context" to mean the situation in which the computation takes place [DA00]. The canonical example of such a situation is the location of the device performing the computation; systems sensitive to location are called *location-aware*. As an example, a location-aware printing system could send a user's print job to a printer close by. (For notions of context different from location, refer to [SBG99]; for large-scale practical examples, see [ACH$^+$01].)

To observe changes in the context, context-aware systems typically include a separate context sensing component that maintains a model of the current context. Such models are known as context models [HIR02] or, more specifically, location models [BD05]. The above-mentioned location-aware printing system would need to maintain a model of the context that supports finding the printer closest to a given device. Such models are informal. There are only very few formal models of context-aware computing (refer to [Hen04] for an overview). We point out Context Unity [RJP04]; in spirit, our proposal is somewhat closer to process calculi than Context Unity is. However, bigraphs differ from traditional process calculi in that we get to write our own reaction rules.

In overall terms, our contribution is two-fold.

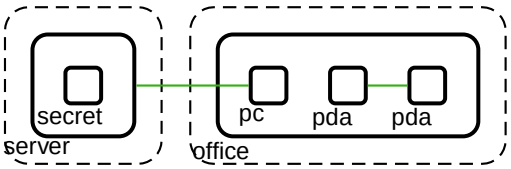
- We find, perhaps surprisingly, that naively modelling context-aware systems as BRSs is somewhat awkward; and
- we propose a more sophisticated modelling technique, in which the perceived and actual context are both explicitly represented as distinct but overlapping BRSs. We call such models *Plato-graphical*.

The remainder of this paper is organised as follows. In section 3.2, we introduce bigraphs and bigraphical reactive systems. In section 3.3, we discuss naive bigraphical models of location-aware systems. In section 3.4, we introduce our Plato-graphical models of context-aware systems. In section 3.5, we present two example models. In section 3.6, we discuss. Finally, in section 3.7, we conclude and note future work.

3.2 Bigraphs and Bigraphical Reactive Systems

We introduce bigraphs by example (the reader can find the relevant formal definitions of [JM04, Mil05c] in appendix A. Readers acquainted with bigraphs may skip this section.

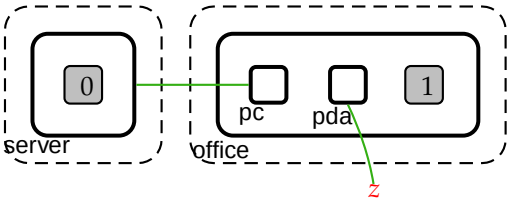
Here is a bigraph, A:



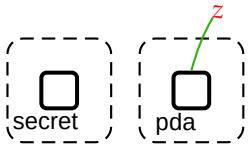
It has *nodes* (vertices), indicated by solid boxes. Each node has a *control*, written in sans serif. Each control has a number of *ports*; ports can be linked by *edges*, indicated by lines. Here, the controls secret and office have no ports, all other controls have one port. Nodes can be nested, indicated by containment. The two outermost dashed boxes indicate *roots*. Roots have no controls; they serve solely to separate different nesting hierarchies.

The bigraph A ostensibly models two physically separate locations (because of the two roots). The first contains a server, which in turn contains secret data; the second contains an office, which in turn contains a PC and two PDAs. The server and the PC are connected, as are the PDAs.

Here is another bigraph, B:



B resembles A, except that the content of server has been replaced with a *site* $-_0$, one of the pdas has been replaced by a site $-_1$, and there is an *inner name* z connected to the remaining pda. Using sites and names, we can define composition of bigraphs. For that, here is yet another bigraph C:



C has an *outer name* z. The bigraphs B and C compose to form A, i.e., $A = B \circ C$. Composition proceeds by plugging the roots of C into the sites of B (in order), and fusing together the connections $\mathsf{pda} \to z$ (in C) and $z \to \mathsf{pda}$ (in B) removing the name z in the process.

One cannot compose arbitrary bigraphs. For $U \circ V$ to be defined, U must have exactly as many sites as V has roots, and the inner names of U must be precisely the outer names of V. The sites and inner names are collectively called the *inner face*; similarly, the roots and outer names are called the *outer face*. A has inner face $\langle 0, \emptyset \rangle$ (no holes, no inner names) and outer face $\langle 2, \emptyset \rangle$ (two roots, no outer names). We write $A : \langle 0, \emptyset \rangle \to \langle 2, \emptyset \rangle$. Similarly, $B : \langle 2, \{z\} \rangle \to \langle 2, \emptyset \rangle$ and $C : \langle 0, \emptyset \rangle \to \langle 2, \{z\} \rangle$.

The graphical representation used above is handy for modelling, but unwieldy for reasoning. Fortunately, bigraphs have an associated term language [DB05, Mil04a], which we use (albeit in a sugared form) in what follows. The language is summarised in Table 3.1. Here are, in order of increasing complexity, term representations of the

Term	Meaning
$U \parallel V$	Concatenation (juxtaposition) of roots.
$U \mid V$	Concatenation (juxtaposition) of children. (collect the children of U and V under one root.)
$U \circ V$	Composition.
$U(V)$	Nesting. U contains V.
$\mathsf{K}_{\vec{x}}(U)$	Ion. Node with control K of arity $\lvert\vec{x}\rvert$, ports connected to the outer names of vector $\vec{x}$. The node contains U.
1	The *barren* (empty) root.
$-_i$	Site numbered i.
$/x.U$	U with outer name x replaced by an edge.
x/y	Connection from inner name y to outer name x.

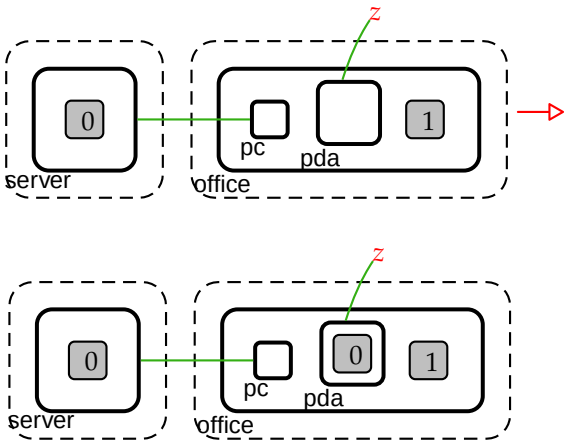
Table 3.1: Sugared term language for bigraphs.

bigraphs A, B and C.

$$
\begin{aligned}
C &= \mathsf{secret} \parallel \mathsf{pda}_z \\
A &= /x./y.\mathsf{server}_x(\mathsf{secret}) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y \mid \mathsf{pda}_y) \\
B &= /x./y.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y \mid -_1) \mid y/z
\end{aligned}
$$

Notice how, in B, edges are specified by first linking nodes to the same name, then converting that name to an edge using the closure '$/$'.

We give dynamics to bigraphs by defining reaction rules. Example:



$$/x.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_z \mid -_1)$$
$$\longrightarrow /x.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_z(-_0) \mid -_1)$$

This rule might model that if a PC in some office is linked to a server, a PDA in the same office may use the PC as a gateway to copy data from the server. The rule matches the bigraph A above, taking secret to the site $-_0$ and pda_y to the site $-_1$, rewriting A to

$$A' = /x./y.\mathsf{server}_x(\mathsf{secret}) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y(\mathsf{secret}) \mid \mathsf{pda}_y)$$

(We omit details on what it means to match connections; refer to one of [JM04, Mil05c].)

It is occasionally convenient to limit the contexts in which a reaction rule applies [BP04], i.e., we might want to limit the above example reaction rule to apply only in the left wing of the building. To this end, bigraphs can be equipped with a *sorting* [Jen06, Mil05c, Mil04b]. A sorting consists of a set of *sorts* (or types); all inner and outer faces are then enriched with such a sort. Further, a sorting must stipulate some condition on bigraphs, we then restrict our attention to the bigraphs that satisfy that condition, thus outlawing some contexts. Obviously, removing contexts may ruin the congruence property of the induced bisimilarity; [Jen06] and [Mil05c] give different sufficient conditions for a sorting to preserve that congruence property.

This concludes our informal overview of bigraphs. Now on to the models.

3.3 Naive Models of Location-aware Systems

In this section, we attempt to model location-aware systems naively in bigraphs. We will find the naive approach to be somewhat awkward. Due to space constraints we do not discuss other forms of context.

We use the place and link graphs for describing locations and interconnections directly, and we use reaction rules to implement both *reconfiguration* of the context and *queries* on the context. The former is simply a non-deterministic change in the context; the latter is a computation on the context that does not change the context, except for producing an answer to some question. In a location-aware system, a device moving would be a reconfiguration, whereas computing the answer to the question "what devices are currently at the location l" is a query.

We discuss the implementation of this query. (An implementation of the query can be found in appendix B.) Incidentally, a query such as "find nearest neighbour", which conceptually is only slightly harder, is significantly harder to implement. (Other examples plagued by essentially the same difficulties can be found in [DD05].)

Consider the following bigraph representing devices (e.g., PDAs) located at locations (e.g., offices, meeting rooms) within a building.

$$l = /w./x./y./z.\mathsf{loc}\Big(\mathsf{loc}\Big(\mathsf{loc}\Big(\mathsf{loc}\,(\mathsf{devi}_w) \mid \mathsf{loc}\Big(\mathsf{devi}_x \mid \mathsf{devi}_y\Big)\Big)\Big) \mid \mathsf{loc}() \mid \mathsf{loc}\,(\mathsf{devi}_z)\Big)$$

Off-hand, finding all devices, say, beneath the root, looks straightforward: We should simply recursively traverse the nesting tree. Unfortunately, such traversal is quite complicated for the following reasons.

- The bigraphical reaction rules do not support recursion directly, so we must encode a runtime stack by means of additional controls.
- Bigraphical reaction rules can be applied in *any* context, but when implementing an operation such as the query we consider now, we need more refined control over when rules can be applied; one may achieve this more refined control by again using additional nodes and controls, essentially implementing what corresponds to a program counter. This still leaves great difficulty in handling concurrent operations, though.
- As a special case of the previous item, it is particularly difficult to express that a reaction rule is intended to apply only in case something is *not* present in the context.

Summing up, the bigraphical rules that model physical action do not in general provide the power to compute directly with a model of that action (because of a lack of control structures). The slogan is "reconfiguring is easy, querying is hard".

In earlier work on evaluating bigraphs as a meta-theory (aim (2) mentioned in the Introduction), reaction rules were used to encode the operational semantics of a calculus or programming language. However, above we attempt to implement a query *directly* as reaction rules. This seemingly innocuous difference will turn out to have major implications for reasoning methods; more on this in subsection 3.6.

We imagine that adding more flexibility to the reaction rules might make it easier to program directly with bigraphs. One possible attempt is to use spatial logics for bigraphs [CMS05] in combination with sorting, to get control of the contexts in which a particular reaction rule applies.

In the following sections, we propose another way to model context-aware systems in bigraphs, where the reaction rules are not used to program directly with, but instead they are used (1) to represent transitions happening in the real world and (2) to encode operational semantics of programming languages, within which one can then implement queries on representations of the real world.

3.4 Plato-graphical Models of Context-aware Systems

The naive model of the previous section shares an important characteristic with recent proposals of formal models for context-aware computation [BP04, NGP05, RJP04] that comprise agents and contexts only: These models take the agent's ability to determine *what is* the present context as given. We contend that for some systems, it is natural to model not only the actual context but also the agent's representation of the actual context. We shall see that pursuing this idea will partially alleviate the awkwardness seen in the previous section.

We shall need some notation and definitions.

NOTATION. We write $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ to indicate that $\mathbf{B}$ is a bigraphical reactive system with controls $\mathcal{K}$ and rules $\mathcal{R}$, and write $f \in \mathbf{B}$ to mean that f is a bigraph of $\mathbf{B}$.

Definition 3.4.1 (Independence). *Let* $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ *and* $\mathbf{B}' = (\mathcal{K}', \mathcal{R}')$ *be bigraphical reactive systems. Say that* $\mathbf{B}$ *and* $\mathbf{B}'$ *are* independent *and write* $\mathbf{B} \perp \mathbf{B}'$ *iff* $\mathcal{K}$ *and* $\mathcal{K}'$ *are disjoint.*

Definition 3.4.2 (Composite bigraphical reactive systems). *Let* $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ *and* $\mathbf{B}' = (\mathcal{K}', \mathcal{R}')$ *be bigraphical reactive systems. Define the union* $\mathbf{B} \cup \mathbf{B}'$ *point-wise, i.e.,* $\mathbf{B} \cup \mathbf{B}' = (\mathcal{K} \cup \mathcal{K}', \mathcal{R} \cup \mathcal{R}')$*, when* $\mathcal{K}$ *and* $\mathcal{K}'$ *agree on the arities of the controls in* $\mathcal{K} \cap \mathcal{K}'$*.*

Be aware that there are two ways of taking the union of two sets of *parametrised* reaction rules: (1) merge the rules and then ground them, or (2) first ground the rules and then merge them. In general, the resulting rule set of (1) is a superset of the rule set of (2). We use approach (1).

We propose a model of context-aware computing that comprises three bigraphical reactive systems: the context $\mathbf{C}$; its representation or proxy $\mathbf{P}$; and the computational agents $\mathbf{A}$. Drawing on classical work [PlaBC], specifically The Allegory of the Cave, we call such a model *Plato-graphical*.

Definition 3.4.3 (Plato-graphical model). *A* Plato-graphical model *is a triple* $(\mathbf{C}, \mathbf{P}, \mathbf{A})$ *of bigraphical reactive systems, such that* $\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$ *is itself a bigraphical reactive system and* $\mathbf{C} \perp \mathbf{A}$*. A* state *of the model is a bigraph of* $\mathcal{M}$ *on the form* $/\vec{x}.(C \parallel P \parallel A)$*, where* $C \in \mathbf{C}$*,* $P \in \mathbf{P}$*,* $A \in \mathbf{A}$*, and* $\vec{x}$ *is some vector of names.*

We emphasise the intended difference between $\mathbf{C}$ and $\mathbf{P}$: Whereas an element of $\mathbf{C}$ models a possible context, an element of $\mathbf{P}$ models *a model* of a possible context. The independence condition ensures that agents can only directly observe or

manipulate the proxy; not the context itself. (In the parlance of [RJP04], the independence condition ensures separability.) To query or alter the context, agents must use the proxy as a sensor and actuator.

Using bigraphs as our basic formalism gives us two things. First, we can write our own reaction rules. We claim that because of this ability, models become remarkably straightforward and intuitive; hopefully, the reader will agree after seeing our example models in the next section. Second, we automatically get a bisimilarity that is a congruence. Thus, bisimilarity of agents is a very fine equivalence: No state of the context and proxy can distinguish bisimilar agents.

Proposition 3.4.1. *Let $\sim$ denote the bisimilarity in $\mathcal{M}$, and let $A, A' \in \mathbf{A}$ with $A \sim A'$. For any $C \in \mathbf{C}$, $P \in \mathbf{P}$, and $\vec{x}$, we have $/\vec{x}.(C \parallel P \parallel A) \sim /\vec{x}.(C \parallel P \parallel A')$.*

To get a less discriminating equivalence we can consider agents under a particular state of the context, or a particular state of the system.

Definition 3.4.4. *Let $\sim$ denote the bisimilarity in $\mathcal{M}$, and let $A, A' \in \mathbf{A}$, $C \in \mathbf{C}$ and $P \in \mathbf{P}$. We say A and A' are* equivalent wrt. *P iff $P \parallel A \sim P \parallel A'$, and we say A and A' are equivalent wrt. C, P iff $C \parallel P \parallel A \sim C \parallel P \parallel A'$.*

We conjecture that the above forms of derived equivalences will prove useful for reasoning about a given Plato-graphical system.

Working within the Plato-graphical model, we are free to emphasise any of its three components, perhaps modelling **P** in great detail, but keeping **C** and **A** abstract.

Definition 3.4.3 above does not impose any restriction on composition of states. For example, assume that we have a Plato-graphical model $\mathcal{M} = (\mathbf{C}, \mathbf{P}, \mathbf{A})$, that c, p and a are controls of **C**, **P** and **A**, respectively, and that p is *not* a control of **C**. Then the bigraphs

$$F = \mathsf{c}(-_0 \mid -_1) \parallel \mathsf{p} \parallel \mathsf{a}(-_2) \qquad \text{and} \qquad G = \mathsf{c} \parallel \mathsf{p} \parallel \mathsf{a}$$

are both states of $\mathcal{M}$, but their composite $F \circ G = \mathsf{c}(\mathsf{c} \mid \mathsf{p}) \parallel \mathsf{p} \parallel \mathsf{a}(\mathsf{a})$ is not a state of $\mathcal{M}$. This example implies that bisimilarity of states of a Plato-graphical system may be too fine a relation: Conceivably, when comparing two states s and s', we may wish to take into account only contexts C such that $C \circ s$ and $C \circ s'$ are themselves states, i.e., we might want to outlaw F as a possible context for G. We can achieve this finer control using place-sorting. So, we define a place-sorted Plato-graphical model. The intuition behind our sorting is that we want to keep controls of **C**, **P** and **A** separate when composing contexts of form $\mathbf{C} \parallel \mathbf{P} \parallel \mathbf{A}$.

NOTATION. Denote by $S_{i\leq m}$ a vector $m_0, \ldots, m_{n-1}$ of sorts. We will write $S_{i\leq m}$ for a sorted interface $\langle m, X, S_{i\leq m}\rangle$ when we do not care about names.

Definition 3.4.5 (Sorted Plato-graphical model). *Let $\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$ be a Plato-graphical model with $\mathbf{C} = (\mathcal{K}_\mathbf{C}, \mathcal{R}_\mathbf{C})$, $\mathbf{P} = (\mathcal{K}_\mathbf{P}, \mathcal{R}_\mathbf{P})$ and $\mathbf{A} = (\mathcal{K}_\mathbf{A}, \mathcal{R}_\mathbf{A})$. Define a sorting discipline on $\mathcal{M}$ by taking sorts $\Theta = \{\mathcal{K}_\mathbf{C}, \mathcal{K}_\mathbf{P}, \mathcal{K}_\mathbf{A}\}$ and, for primes, sorting condition $\Phi(f : S_{i\leq n} \to S) = ctrl(f) \subseteq S \wedge \forall i \leq n.\ S_i = S$, lifting to an arbitrary*

bigraph f' by decomposing f into primes $f' = f_0 \dots f_{n-1}$ and declaring f' well-sorted iff all the f_i are. Let ϕ be an assignment of Θ-sorts to the rules of $\mathcal{R}_{\mathbf{C}}$, $\mathcal{R}_{\mathbf{P}}$, and $\mathcal{R}_{\mathbf{A}}$, such that every rule is well-sorted under Φ. Define $\mathcal{M}'$ to be $\mathcal{M}$ sorted by (Θ, Φ) (using ϕ to lift the reaction rules). In this case, we call $\mathcal{M}'$ a sorted Plato-graphical model, *and define the states of $\mathcal{M}'$ to be the well-sorted bigraphs with outer face $\mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{P}}, \mathcal{K}_{\mathbf{A}}$.*

The condition Φ essentially requires that (1) the controls of a prime (bigraph) are elements of the sort of its outer face, and (2) the sort of the outer face is exactly the sort of each of the sites. Under this sorting discipline and new definition of state, if G is assigned a sort such that it is a state, then F cannot be assigned a sort that makes it composible with G.

Is the bisimilarity in the sorted system $\mathcal{M}'$ a congruence? The sorting discipline of $\mathcal{M}'$ is in general not homomorphic in the sense of Milner [Mil05c, Definition 10.4]: we cannot give a sort to controls in $\mathcal{K}_{\mathbf{C}} \cap \mathcal{K}_{\mathbf{P}}$. (If **C**, **P** and **A** are pairwise independent, the sorting is homomorphic; however, such a model is pathologic.) Neither is the sorting safe in the sense of Jensen [Jen06, Definition 4.30]; condition (4) cannot be met. Counterexample: Suppose $f : \mathcal{K}_{\mathbf{C}} \to \mathcal{K}_{\mathbf{C}}$ is well-sorted; take $g = f \otimes \mathsf{1} : \mathcal{K}_{\mathbf{C}} \to \mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{A}}$ (recall that $\mathsf{1} : \epsilon \to \langle 1, \emptyset \rangle$ denotes the barren root). Clearly, $\mathcal{U}(f) = (-_0 \mid -_1) \circ \mathcal{U}(f \otimes \mathsf{1})$. However, if $\mathcal{K}_{\mathbf{C}} \neq \mathcal{K}_{\mathbf{A}}$ then $(-_0 \mid -_1) : \mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{A}} \to \mathcal{K}_{\mathbf{C}}$ is not well-sorted.

Nevertheless, the sorting of definition 3.4.5 does give rise to a bisimilarity that is a congruence; we prove so in Appendix C.

3.5 Examples

3.5.1 A Simple Context-aware Printing System

We model the simple context-aware printing system of [BP04]. An office-building contains both modern PCL-5e compatible printers and old-fashioned raw-printers. Occasionally, the IT-staff at the building removes or replaces either type of printers. Each printer can process only one job; queueing is done by a central print server. The print server dispatches jobs to raw-printers only if it knows no PCL-printers; if there are PCL-printers, but they are all busy, the job will simply have to wait. This system is context-aware: The type and number of printers physically available determine the meaning of the action "to print". We give a model **B** of this system in Figure 3.1. Looking at the controls of **B**, it is straightforward to verify that **B** is Plato-graphical.

Proposition 3.5.1. *The model* **B** *of Figure 3.1 is Plato-graphical.*

We take a detailed look at the model. A state of the context **C** consists of nested physical locations loc, within which printers prt are placed. We distinguish between PCL- and raw-printers by putting a token pcl and raw within them, respectively. Each printer has a single port, intended to link the printer to the proxy. Here is a

Context **C**.

Control	Activity	Arity	Comment
loc	active	0	Nested location
prt	passive	1	Physical printer
pcl	atomic	0	Printer-type token
raw	atomic	0	Printer-type token
dat	atomic	1	Binary data for printer

$$\mathsf{loc}(-_0) \longrightarrow \mathsf{loc}(-_0 \mid /x.\mathsf{prt}_x(\mathsf{raw})) \tag{3.1}$$

$$\mathsf{loc}(-_0) \longrightarrow \mathsf{loc}(-_0 \mid /x.\mathsf{prt}_x(\mathsf{pcl})) \tag{3.2}$$

$$\mathsf{loc}(-_0 \mid \mathsf{prt}_x(-_1)) \longrightarrow \mathsf{loc}(-_0) \mid x/ \tag{3.3}$$

$$\mathsf{prt}_x(\mathsf{dat}_z \mid -_0) \longrightarrow \mathsf{prt}_x(-_0) \mid z/ \tag{3.4}$$

Proxy **P**.

Control	Activity	Arity	Comment
prt	passive	1	Physical printer
pcl	atomic	0	Printer-type token
raw	atomic	0	Printer-type token
dat	atomic	1	Binary data for printer
prts	passive	1	Known devices
jobs	passive	0	Pending documents
doc	atomic	1	Document

$$\begin{aligned}\mathsf{jobs}(\mathsf{doc}_z \mid -_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{prt}_y(\mathsf{pcl}) &\longrightarrow \\ \mathsf{jobs}(-_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) &\parallel \mathsf{prt}_y(\mathsf{pcl} \mid \mathsf{dat}_z)\end{aligned} \tag{3.5}$$

$$\begin{aligned}\mathsf{jobs}(\mathsf{doc}_z \mid -_0) \parallel /x.\mathsf{prts}_x(\mathsf{pcl}) \mid \mathsf{prts}_y(\mathsf{raw}) \parallel \mathsf{prt}_y(\mathsf{raw}) &\longrightarrow \\ \mathsf{jobs}(-_0) \parallel /x.\mathsf{prts}_x(\mathsf{pcl}) \mid \mathsf{prts}_y(\mathsf{raw}) &\parallel \mathsf{prt}_y(\mathsf{raw} \mid \mathsf{dat}_z)\end{aligned} \tag{3.6}$$

$$/x.\mathsf{prt}_x(\mathsf{pcl}) \parallel \mathsf{prts}_y(\mathsf{pcl}) \longrightarrow \mathsf{prt}_y(\mathsf{pcl}) \parallel \mathsf{prts}_y(\mathsf{pcl}) \tag{3.7}$$

$$/x.\mathsf{prt}_x(\mathsf{raw}) \parallel \mathsf{prts}_y(\mathsf{raw}) \longrightarrow \mathsf{prt}_y(\mathsf{raw}) \parallel \mathsf{prts}_y(\mathsf{raw}) \tag{3.8}$$

Agents **A**.

Control	Activity	Arity	Comment
jobs	passive	0	Pending documents
doc	atomic	1	Document

$$\mathsf{jobs}(-_0) \longrightarrow \mathsf{jobs}(-_0 \mid /z.\mathsf{doc}_z) \tag{3.9}$$

Figure 3.1: Example Plato-graphical model **B**.

Context **C**	Proxy **P**	Agent **A**
(3.1) : $\mathcal{K}_{\mathbf{C}}$	(3.5) : $\mathcal{K}_{\mathbf{A}}, \mathcal{K}_{\mathbf{P}}, \mathcal{K}_{\mathbf{C}}$	(3.9) : $\mathcal{K}_{\mathbf{A}}$
(3.2) : $\mathcal{K}_{\mathbf{C}}$	(3.6) : $\mathcal{K}_{\mathbf{A}}, \mathcal{K}_{\mathbf{P}}, \mathcal{K}_{\mathbf{C}}$	
(3.3) : $\mathcal{K}_{\mathbf{C}}$	(3.7) : $\mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{P}}$	
(3.4) : $\mathcal{K}_{\mathbf{C}}$	(3.8) : $\mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{P}}$	

Figure 3.2: Sorts for the rules of **C**, **P**, and **A**.

state of the context with a PCL-printer and a raw-printer at adjacent locations; the PCL-printer is idle whereas the raw-printer is busy.

$$C = \mathsf{loc}(\mathsf{loc}(\mathsf{prt}_x(\mathsf{raw} \mid \mathsf{dat}_z)) \mid \mathsf{loc}(/y.\mathsf{prt}_y(\mathsf{pcl})))$$

Setting C in parallel with some proxy P will allow P access to the raw printer through the shared link x, but not to the PCL-printer, because it is in a closed link. The dynamics of **C** allow printers to appear (3.1, 3.2), disappear (3.3), and finish printing (3.4).

A state of the proxy **P** consists of a pool of pending jobs jobs and two tables of printers prts; one contains a token raw, the other a token pcl, indicating what type of printer the table lists. The prts is a table in the sense that its only port is linked to all the printers in the context that the table knows about. Here is an example state of the proxy which knows one raw-printer, knows no PCL-printers and has two pending jobs.

$$P = \mathsf{prts}_x(\mathsf{raw}) \mid /y.\mathsf{prts}_y(\mathsf{pcl}) \mid \mathsf{jobs}(/z.\mathsf{doc}_z \mid /z'.\mathsf{doc}_{z'})$$

Setting C and P above in parallel by $\|$, and closing the link x, we get a system $/x.C \parallel P$, where the table $\mathsf{prts}_x(\mathsf{raw})$ and the physical printer $\mathsf{prt}_x(\mathsf{raw} \mid \mathsf{dat})$ are linked. The dynamics of **P** state that if there is a job and a known, idle PCL-printer, the proxy may activate this printer (3.5); that if there is a job, no known PCL-printer, and an idle raw-printer, the context may activate that printer (3.6); and finally, that the proxy may discover a previously unknown printer (3.7, 3.8).

The dynamics of **A** allow the agents to spontaneously spool documents (3.9).

Notice how the two printing rules (3.5) and (3.6) do not observe the context directly. Instead, the proxy observes the context (rules (3.7) and (3.8)) and records its observations in the tables $\mathsf{prts}_x(\mathsf{raw})$ and $\mathsf{prts}_y(\mathsf{pcl})$; the printing rules (3.5) and (3.6) then consults the tables. It is straightforward to determine whether there are no known PCL-printers: simply check if the table of PCL-printers has the form $/y\,.\,\mathsf{prts}_y(\mathsf{pcl})$.

As observed in section 3.3 and [BP04], it is generally very difficult, if not impossible, to observe the *absence* of something in the context directly. An interesting but rather natural consequence of the indirect observation is that it becomes asynchronous, i.e., it is possible that a PCL-printer exists but has not yet been observed.

This model **B** can be lifted to a sorted one by adding the sorts given in Figure 3.2; the figure assigns sorts to the outer face of both the redeces and reactums of the indicated rules. It is straightforward to verify that all of the rules are well-sorted.

Proposition 3.5.2. *The model* **B** *with the sorting assignment of Figure 3.2 is a sorted Plato-graphical model.*

3.5.2 A Location-aware Printing System

Suppose we extend the printing system with location-awareness, by stipulating that a print job is not printed until the printer and the device submitting the job are co-located. To model this extended system, we introduce a new control devi for devices (PCs or PDAs) with one port and change doc to include an extra port so we can link submitted jobs to the devices submitting them. The linking is reflected in the following modified rule (3.9) for spooling print jobs:

$$\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{jobs}(-_1) \longrightarrow \mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{jobs}(-_1 \mid /z.\mathsf{doc}_{z,x}) \quad (3.9')$$

We must also modify rules (3.5) and (3.6) to insist that the device and printer are co-located. Rule (3.5) becomes

$$\begin{aligned} \mathsf{jobs}(\mathsf{doc}_{z,x} \mid -_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{prt}_y(\mathsf{pcl})) \longrightarrow \\ \mathsf{jobs}(-_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{prt}_y(\mathsf{pcl} \mid \mathsf{dat}_z)). \end{aligned} \quad (3.5')$$

(We suppress the new Rule (3.6').)

Modifying the system once again, instead of insisting that device and printer have to be actually co-located, we just require the print job to end at a printer close to the device. The print server will need to query the proxy for the printer nearest a given device. We saw in subsection 3.3 that implementing such queries is awkward, so we will need to use the proxy. In the preceding section, we did so directly in bigraphs; this time around, we transfer the expressive convenience of a general-purpose programming language to bigraphs for ease of implementation. We use bigraphs directly for modelling the actual context **C**, whereas we will exploit bigraphs as a meta-calculus for modelling the proxy **P**.

In detail, the whole model is $\mathbf{B} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$, with $\mathbf{P} = \mathbf{S} \cup \mathbf{L}$. Here **C** is intended to be a bigraphical model of the "real world", the proxy **P** is comprised of a location sensor **S** and a location model **L** and **A** is the location-based application (the "computational agent").

A state C of **C** could look like this:

$$C = \mathsf{loc}(\mathsf{loc}(\mathsf{loc}(\mathsf{loc}(\mathsf{devi}_w) \mid \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{devi}_y))) \mid \mathsf{loc} \mid \mathsf{loc}(\mathsf{devi}_z))$$

Changes in the real world are modeled by reaction rules that reconfigure such states. If we want to model, say, that a devices may move from one location to another, we include the reaction rule

$$\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{loc}(-_1) \longrightarrow \mathsf{loc}(-_0) \parallel \mathsf{loc}(\mathsf{devi}_x \mid -_1). \quad (3.10)$$

To implement the proxy, encode as a BRS a programming language $\mathcal{L}$ with data structures, communication primitives, and concurrency, e.g., Pict [PT00] or CML

[Rep99]. (We return to this assumption below.) That is, define a translation from terms of $\mathcal{L}$ to bigraphs, and add reaction rules encoding the operational semantics of $\mathcal{L}$. *Then implement the location model, the sensor, and the agents in $\mathcal{L}$ and use the encoding to transfer that model to bigraphs.* In particular, a state of the location model **L** will have a data structure *representing* the current state of **C**. If $\mathcal{L}$ is an even half-way decent programming language, it should be straightforward to implement queries such as one of section 3.3 or the "find closest printer" we need above.

The sensor informs the location model about changes in **C**. We extend the above rule (3.10) moving a device to

$$(\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{loc}(-_1)) \mid S \mid L \longrightarrow (\mathsf{loc}(-_0) \parallel \mathsf{loc}(\mathsf{devi}_x \mid -_1)) \mid S' \mid L, \quad (3.10')$$

where S' is an $\mathcal{L}$-encoding of "send a notification to **L** that device x has moved". Upon receiving the notification, **L** updates its representation of the world. Agents of **A** can in turn query **L** when they need location information.

3.6 Discussion

We consider the following questions.

1. What languages $\mathcal{L}$ can we encode?
2. How close are Plato-graphical models to real systems?
3. What challenges have we found for bigraphical models?
4. What uses do we envision for Plato-graphical models?
5. How do we reason about Plato-graphical models?

Ad. 1. As mentioned, there exist bigraphical encodings of various π-calculi [Jen06, JM04, JM03] and of the λ-calculus [Mil04c, Mil05b]. Using ideas of the latter encodings, we have encoded Mini-ML (call-by-value λ-calculus with pairs and lists) in local bigraphs [Mil04c]. Based on our experiences with this encoding, we find it palatable to encode CML or Pict[1].

Ad. 2. The model closely reflects how some actual location-aware systems work, for instance the one running at the ITU. Here, a sensor system (made by Ekahau) computes every two seconds the physical location of every device on the WLAN. The sensor system informs a location model about updates to locations; location-aware services then interact with the location model. In our sketched Plato-graphical model, the location model **L** may lag behind the actual **C**, if **L**'s representation of **C** does not reflect some recent reconfiguration of **C**. But that also happens in the real system at the ITU – when a location-aware service asks the location model for the whereabouts of a device, it obtains not the position of the device, but the position of

[1] We are presently working on implementing an interpreter for bigraphical reactive systems; such an interpreter will make it easier to experiment with these and other encodings.

the device the last time the sensor checked. In the mean time, the device may have moved.

Ad. 3. When modelling the physical world, we have made use of both the place and link graphs, the place graph modelling the location hierarchy of a building. As argued in [BD05], DAGs or graphs are more natural models of location. Thus, systems such as the ones we have considered here suggest generalising the place graph part of bigraphs, or consider ways to encode DAGs or general graphs naturally as place graphs.

Ad. 4. Given an implementation of bigraphical reactive systems, one could *simulate* the behaviour of a location-aware system, and thus allow for experimentation with different designs of location-aware and context-aware systems. Likewise, one could experiment with different choices for the $\mathcal{L}$ language of section 3.5.2. Such simulation suggests further extensions of the bigraphical model: In actual context-aware systems, one is generally interested in timing aspects (e.g., the sensor samples only every two seconds), continuous space (e.g., the sensor really produces continuous data), and probabilistic models (e.g., to accurately simulate sensors and sensor failure).

Ad. 5. What about using Plato-graphical models for *formal reasoning* about context-aware systems? One use of formal models is to prove an abstract specification model equivalent to a concrete implementation model. In π-calculus, we come with π-terms i, s, one for the implementation and one for the specification. The terms i and s are themselves the models; we take (π-) bisimilarity as equivalence, so to prove i and s equivalent, we merely prove them bisimilar. We can play the same game within any BRS: Simply come up with a bigraph I (the implementation model) and a bigraph S (the specification model), and prove them bisimilar within the labelled transition system of the BRS. Because that bisimulation is a congruence, such reasoning should be tractable, e.g. with the bisimulation in definition 3.4.4.

Unfortunately, bisimulation within a single BRS is not always enough wrt. Plato-graphical models. Suppose we want a specification model $\mathcal{M}$ with an abstract view of the context, and an implementation model $\mathcal{M}'$ with a detailed view of the context. We express this by having $\mathcal{M}$ and $\mathcal{M}'$ differ only in their context sub-BRSs, that is,

$$\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A} \qquad \mathcal{M}' = \mathbf{C}' \cup \mathbf{P} \cup \mathbf{A}.$$

The trouble is that because $\mathbf{C}$ and $\mathbf{C}'$ may have different controls and reaction rules, bisimulation between their respective labelled transition systems is meaningless! What we need is a notion of equivalence of BRSs, not just equivalence of bigraphs of a single BRS. At the time of writing, we know of no such equivalence[2]. Thus, our investigation of bigraphical models for context-aware systems suggests that equivalence of BRSs is a key notion currently missing. One possible direction would be

[2]The reader may suggest that we just define a common language for modelling both the abstract and detailed view, and define a translation from this language into a single BRS. However, in this case we are no longer modelling a ubiquitous system directly in bigraphs (aim 1 of the Introduction), but using bigraphs as a meta-calculus (aim 2 of the Introduction).

to try to recover from the notion of WRS-functor [LM00] – functors that preserve reaction rules – a notion of a BRS implementing another BRS.

3.7 Conclusion & Future Work

We have initiated an evaluation of the use of bigraphical reactive systems for models of context-aware computing in ubiquitous systems. We found that BRSs, in their current form, are not suitable for directly modelling context queries, but on the other hand suitable for modelling reconfigurations of the actual context.

In response, we proposed Plato-graphical models, where both state and dynamics are logically divided in three parts: the actual context, the observed context (or proxy), and the computational agents, respectively. The computational agents and the actual context are separated, and interact only through the proxy. This separation into different BRSs makes it possible to encode different parts of the system using domain-specific languages. Moreover, we have shown how the context-aware printing system of [BP04] can be modeled straightforwardly in the Plato-graphical model.

Further, we have argued that Plato-graphical models are useful for simulating context-aware systems, and we are currently working on an implementation of BRSs at ITU to allow such experimentation. Only through such experimentation will it be clear how useful Plato-graphical models really are. For simulation purposes it will be important to extend bigraphs with timing aspects, continuous space, and probabilities.

Finally, we have pointed out that establishing a notion of equivalence between BRSs, as opposed to bisimilarity within a BRS, is important future work.

3.8 Acknowledgements

We gratefully acknowledge discussions with the other members of the BPL group at ITU, in particular Arne Glenstrup, Troels Damgaard and Mikkel Bundgaard; and with Robin Milner. This work was funded in part by the Danish Research Agency (grant no.: 2059-03-0031) and the IT University of Copenhagen (the LaCoMoCo project).

A Bigraphs

We recite the identical relevant definitions of [JM04] and a few from [Mil05c].

Definition A.1 (pure signature). *A* (pure) signature $\mathcal{K}$ *is a set whose elements are called* controls*. For each control* K *it provides a finite ordinal* $ar(K)$*, an* arity*; it also determines which controls are* atomic*, and which of the non-atomic controls are* active*. Controls which are not active (including the atomic controls) are called* passive*.*

Definition A.2 (prime interface). *An* interface $I = \langle m, X \rangle$ *consists of a finite ordinal* m *called a* width, *a finite set* X *called a* name set. *An interface is* prime *if it has width* 1.

Definition A.3 (prime bigraph). *A* prime *bigraph* $P : m \to \langle X \rangle$ *has no inner names and a prime outer face.*

Definition A.4 (place graph). *A* place graph $A = (V, ctrl, prnt) : m \to n$ *has an* inner width m *and an* outer width n, *both finite ordinals; a finite set* V *of nodes with a control map* $ctrl : V \to \mathcal{K}$; *and a* parent map $prnt : m \uplus V \to V \uplus n$. *The parent map is* acyclic, *i.e.* $prnt^k(v) \neq v$ *for all* $k > 0$ *and* $v \in V$. *An* atomic *node – i.e. one whose control is atomic – may not be a parent. We write* $w >_A w'$, *or just* $w > w'$, *to mean* $w = prnt^k(w')$ *for some* $k > 0$.

The widths m *and* n *index the* sites *and* roots *of* A *respectively. The sites and nodes – i.e. the domain of* $prnt$ *– are called* places.

Definition A.5 (precategory of place graphs). *The precategory of place graphs* ´PLG *has finite ordinals as objects and place graphs as arrows. The composition* $A_1 \circ A_0 : m_0 \to m_2$ *of two place graphs* $A_i = (V_i, ctrl_i, prnt_i) : m_i \to m_{i+1}$ $(i = 0, 1)$ *is defined when the two node sets are disjoint; then* $A_1 \circ A_0 \stackrel{def}{=} (V, ctrl, prnt)$ *where* $V = V_0 \uplus V_1$, $ctrl = ctrl_0 \uplus ctrl_1$, *and* $prnt = (\mathsf{Id}_{V_0} \uplus prnt_1) \circ (prnt_0 \uplus \mathsf{Id}_{V_1})$. *The identity place graph at* m *is* $\mathsf{id}_m \stackrel{def}{=} (\emptyset, \emptyset_{\mathcal{K}}, \mathsf{Id}_m) : m \to m$.

Definition A.6 (tensor product, ´PLG). *The* tensor product $\otimes$ *in* ´PLG *is defined as follows: On objects, we take* $m \otimes n = m + n$. *For two place graphs* $A_i : m_i \to n_i$ $(i = 0, 1)$ *we take* $A_0 \otimes A_1 : m_0 + m_1 \to n_0 + n_1$ *to be defined when* A_0 *and* A_1 *have disjoint node sets; for the parent map, we first adjust the sites and roots of* A_1 *by adding them to* m_0 *and* n_0 *respectively, then take the union of the two parent maps.*

Definition A.7 (barren,sibling,active,passive). *A node or root is* barren *it is has no children. Two places are* siblings *if they have the same parent. A site* s *of* A *is* active *if* $ctrl(v)$ *is active whenever* $v > s$; *otherwise* s *is* passive. *If* s *is active (resp. passive) in* A, *we also say that* A *is* active *(resp.* passive*)* at s.

Definition A.8 (hard place graphs). *A* hard *place graph is one in which no root or non-atomic node is barren. They form a sub-precategory denoted by* ´$\mathrm{PLG_h}$.

Presuppose a denumerable set χ of global names.

Definition A.9 (link graph). *A* link graph $A = (V, E, ctrl, link) : X \to Y$ *has finite sets* X *of* inner names, Y *of* (outer) names, V *of* nodes *and* E *of* edges. *It also has a function* $ctrl : V \to \mathcal{K}$ *called the* control map, *and a function* $link : X \uplus P \to E \uplus Y$ *called the* link map, *where* $P \stackrel{def}{=} \sum_{v \in V} ar(ctrl(v))$ *is the set of* ports *of* A.

We shall call the inner names X *and ports* P *the* points *of* A, *and the edges* E *and outer names* Y *its* links.

Definition A.10 (precategory of link graphs). *The precategory* ´LIG *has name sets as objects and link graphs as arrows. The composition* $A_1 \circ A_0 : X_0 \to X_2$ *of two link graphs* $A_i = (V_i, E_i, ctrl_i, link_i) : X_i \to X_{i+1}$ $(i = 0, 1)$ *is defined when their node sets and edge sets are disjoint; then* $A_1 \circ A_0 \stackrel{def}{=} (V, E, ctrl, link)$ *where* $V = V_0 \uplus V_1$, $ctrl = ctrl_0 \uplus ctrl_1$, $E = E_0 \uplus E_1$ *and* $link = (\mathsf{Id}_{E_0} \uplus link_1) \circ (link_0 \uplus \mathsf{Id}_{P_1})$. *The identity link graph at* X *is* $\mathsf{id}_X = (\emptyset, \emptyset, \emptyset_{\mathcal{K}}, \mathsf{Id}_X) : X \to X$.

Definition A.11 (tensor product, ´LIG). *The* tensor product $\otimes$ *in* ´LIG *is defined as follows: On objects,* $X \otimes Y$ *is simply the union of sets required to be disjoint. For two link graphs* $A_i : X_i \to Y_i$ $(i = 0, 1)$ *we take* $A_0 \otimes A_1 : X_0 \otimes X_1 \to Y_0 \otimes Y_1$ *to be defined when the interface products are defined and when* A_0 *and* A_1 *have disjoint node sets and edge sets; then we take the union of their link maps.*

Definition A.12 (parallel product). *The* parallel product $\|$ *in* ´LIG *is defined as follows: On objects,* $X \parallel Y \stackrel{def}{=} X \cup Y$. *On link graphs* $A_i : X_i \to Y_i$ $(i = 0, 1)$ *we define* $A_0 \parallel A_1 : X_0 \otimes X_1 \to Y_0 \parallel Y_1$ *whenever* X_0 *and* X_1 *are disjoint, by taking the union of link maps.*

Definition A.13 (concrete pure bigraph). *A* (concrete) pure bigraph *over the signature* $\mathcal{K}$ *takes the form* $G = (V, E, ctrl, G^{\mathsf{P}}, G^{\mathsf{L}}) : I \to J$ *where* $I = \langle m, X \rangle$ *and* $J = \langle n, Y \rangle$ *are its* inner *and* outer faces, *each combining a* width *(a finite ordinal) with a finite set of global names drawn from* χ. *Its first two components* V *and* E *are finite sets of* nodes *and* edges *respectively. The third component* $ctrl : V \to \mathcal{K}$, *a* control map, *assigns a control to each node. The remaining two are:* $G^P = (V, ctrl, prnt) : m \to n$, $G^L = (V, E, ctrl, link) : X \to Y$.

A place graph can be combined with a link graph iff they have the same node set and control map.

Definition A.14 (Tensor product). *The* tensor product *of two bigraph interfaces is defined by* $\langle m, X \rangle \otimes \langle n, Y \rangle \stackrel{def}{=} \langle m + n, X \uplus Y \rangle$ *when* X *and* Y *are disjoint. The* tensor product *of two bigraphs* $G_i : I_i \to J_i (i = 0, 1)$ *is defined by* $G_0 \otimes G_1 \stackrel{def}{=} \langle G_0^P \otimes G_1^P, G_0^L \otimes G_1^L \rangle : I_o \otimes I_1 \to J_0 \otimes J_1$ *when the interfaces exist and the node sets are disjoint. This combination is well-formed, since its constituents share the same node set.*

Definition A.15 (precategory of pure concrete bigraphs). *The precategory* ´BIG$(\mathcal{K})$ *of pure concrete bigraphs over a signature* $\mathcal{K}$ *has pairs* $I = \langle m, X \rangle$ *as objects (*interfaces*) and bigraphs* $G = (V, E, ctrl_G, G^{\mathsf{P}}, G^{\mathsf{L}}) : I \to J$ *as arrows (*contexts*). We call* I *the* inner face *of* G, *and* I *the* outer face. *If* $H : J \to K$ *is another bigraph with node set disjoint from* V, *then their composition is defined directly in terms of the compositions of the constituents as follows:* $H \circ G \stackrel{def}{=} \langle H^{\mathsf{P}} \circ G^{\mathsf{P}}, H^{\mathsf{L}} \circ G^{\mathsf{L}} \rangle : I \to K$. *The identities are* $\langle \mathsf{id}_m, \mathsf{id}_X \rangle : I \to I$, *where* $I = \langle m, X \rangle$.

The subprecategory ´BIG$_{\mathsf{h}}$ *consists of* hard *bigraphs, those with place graphs in* ´PLG$_{\mathsf{h}}$.

Definition A.16 (tensor product, ´BIG). *The* tensor product *of two bigraph interfaces is defined by* $\langle m, X\rangle \otimes \langle n, Y\rangle \stackrel{def}{=} \langle m+n, X \cup Y\rangle$ *when* X *and* Y *are disjoint. The* tensor product *of two bigraphs* $G_i : I_i \to J_i$ $(i = 0, 1)$ *is defined by* $G_0 \otimes G_1 \stackrel{def}{=} \langle G_0^{\mathsf{P}} \otimes G_1^{\mathsf{P}}, G_0^{\mathsf{L}} \otimes G_1^{\mathsf{L}}\rangle : I_0 \otimes I_1 \to J_0 \mid J_1$ *when the interfaces exist and the node sets are disjoint. This combination is well-formed, since its constituents share the same node set.*

Definition A.17 (parallel product, ´BIG). *The* parallel product *of two bigraphs is defined on interfaces by* $\langle m, X\rangle \mid \langle n, Y\rangle \stackrel{def}{=} \langle m+n, X \cup Y\rangle$, *and on bigraphs by* $G_0 \mid G_1 \stackrel{def}{=} \langle G_0^P \otimes G_1^P, G_0^L \parallel G_1^L\rangle : I_o \otimes I_1 \to J_0 \mid J_1$ *when the interfaces exist and the node sets are disjoint.*

Refer to [JM04] for the definition of ´BBG, ´BBG$_{\mathsf{h}}$, and $\doteq$.

Definition A.18 (bigraphical reactive system). *A* bigraphical reactive system (BRS) *over* $\mathcal{K}$ *consists of* ´BBG($\mathcal{K}$) *equipped with a set* ´Reacts *of reaction rules closed under support equivalence (*$\doteq$*). We denote it – and similarly for* ´BBG$_{\mathsf{h}}$($\mathcal{K}$) *– by* ´BBG($\mathcal{K}$, ´Reacts).

Refer to [Mil05c] for the definition of *s-category*.

Definition A.19 (place-sorted bigraphs). *An interface* $\langle m, X\rangle$ *is* Θ-(place-)sorted *if it is enriched by ascribing a sort to each place* $i \in m$. *If* I *is place-sorted we denote its underlying unsorted interface by* $\mathcal{U}(I)$.

We denote by ´BIG$_{\mathsf{h}}$($\mathcal{K}, \Theta$) *the s-category in which the objects are place-sorted interfaces, and each arrow* $G : I \to J$ *is a bigraph* $G : \mathcal{U}(I) \to \mathcal{U}(J)$. *The identities, composition and tensor product are as in* ´BIG$_{\mathsf{h}}$($\mathcal{K}$), *but with sorted interfaces.*

Definition A.20 (place-sorting). *A* place-sorting *is a triple* $\Sigma = (\mathcal{K}, \Theta, \Phi)$ *where* Φ *is a condition on the place graphs of* Θ*-sorted bigraphs over* $\mathcal{K}$. *The condition* Φ *must be satisfied by the identities and preserved by composition and tensor product.*

A bigraph in ´BIG$_{\mathsf{h}}$($\mathcal{K}, \Theta$) *is* Σ(-place)sorted *if it satisfies* Φ. *The* Σ*-sorted bigraphs form a sub-s-category of* ´BIG$_{\mathsf{h}}$($\mathcal{K}, \Theta$) *denoted by* ´BIG$_{\mathsf{h}}$(Σ). *Further, if* ´$\mathcal{R}$ *is a set of* Σ*-sorted reaction rules then* ´BIG$_{\mathsf{h}}$(Σ, ´$\mathcal{R}$) *is a* Σ*-sorted BRS.*

B Encoding of "find all devices"

Consider the following simple bigraph representing a building consisting of locations (e.g., rooms) and devices (e.g., PDAs) in these locations. (We have omitted the outer names on the locations, and also sites.)

$$l = \mathsf{loca}(\mathsf{loca}(\mathsf{loca}(\mathsf{loca}(\mathsf{devi}_1) \mid \mathsf{loca}(\mathsf{devi}_2 \mid \mathsf{devi}_3))) \mid \mathsf{loca}() \mid \mathsf{loca}(\mathsf{devi}_4))$$

Consider how to implement a query to return all the devices in the building by means of bigraphical reaction rules. Observe that we have chosen to represent all locations via the same control loca, rather than using different controls office, building, etc.,

for different locations – this is to avoid having to write reaction rules for every combination of location controls.

Now, assume that a query occurs by some process introducing a node with control f into the system (in a unique[3] in node), and that no other queries impose themselves while we calculate the answer to this one. The termination condition (observable by the "input/output process") is when in is empty (and nodes with control f′ appear in the node with unique control out). We can not handle concurrent queries so that is why we wish to detect termination (so that we can begin the next query).

The idea is to do a depth-first search/collection while keeping track of where we have already looked by placing these subtrees into "searched-nodes" (s). Controls:

Control	Activity	Arity	Comment
in	passive	0	Input node
f	atomic	0	Controls find-all query
f′	atomic	1	Answer node
loca	active	1	Nested location
out	passive	0	Output node
g	atomic	0	Dummy, just to keep in non-empty
devi	atomic	1	Device, has link to id
s	passive	0	Collects searched nodes

And now, for the rules. Initialization; move f into the *top* location (enclosing all the others) to indicate "the point of control" in the structure, and add g to indicate that we are not done with the query:

$$\mathsf{in}(\mathsf{f}) \mid \mathsf{loca}_{top}(\mathsf{loca}_x(-_0)) \mid \mathsf{out}() \longrightarrow \mathsf{in}(\mathsf{g}) \mid \mathsf{loca}_{top}(\mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}() \mid -_0)) \mid \mathsf{out}()$$

If a device is found, add it to s, and add a representative for it to out:

$$\begin{aligned} & \mathsf{loca}_x(\mathsf{f} \mid -_0 \mid \mathsf{devi}_y \mid \mathsf{s}(-_1)) \mid \mathsf{out}(-_2) \\ \longrightarrow\ & /y.\mathsf{loca}_x(\mathsf{f} \mid -_0 \mid \mathsf{s}(-_1 \mid \mathsf{devi}_y)) \mid \mathsf{out}(-_2 \mid \mathsf{f}'_y) \end{aligned}$$

Notice that the label (context) of this transition will include the bigraph top/x. This rule can be used as long as there are devices in the current location being searched. When done with this location f is moved up, since we assume that a location can only contain either devices or other locations. (The query would have been significantly harder without this assumption.) So, if a location containing location(s) is being searched:

$$\mathsf{loca}_x(\mathsf{f} \mid \mathsf{loca}_y(-_1) \mid \mathsf{s}(-_2) \mid -_0) \longrightarrow \mathsf{loca}_x(\mathsf{loca}_y(\mathsf{f} \mid \mathsf{s}() \mid -_1) \mid \mathsf{s}(-_2) \mid -_0)$$

Then, search deeper. A new s-node is created when going down, this is a trick to do "on-line garbage collection" when climbing back up the tree. (The query has to leave the bigraph as it was initially.) When a leaf is reached (an empty location) move f up,

[3] Certain properties can only be ensured by invariants of the reactive system, e.g. uniqueness of controls.

merging s-nodes. This is more clever than doing a clean-up traversal because there is no construction in reaction rules that can express "not", i.e. one can not write a rule saying "clean up until there are no more s-nodes in the tree". The "climbing" rule:

$$\mathsf{loca}_x(\mathsf{loca}_y(\mathsf{f} \mid \mathsf{s}(-_2)) \mid \mathsf{s}(-_1) \mid -_0) \longrightarrow \mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}(\mathsf{loca}_y(-_2) \mid -_1) \mid -_0)$$

Notice how the presented rules use sites to make themselves general. To clean up when we have traversed the whole tree (and there is exactly one s):

$$\mathsf{in}(g) \mid \mathsf{loca}_{top}(\mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}(-_0))) \longrightarrow \mathsf{in}() \mid \mathsf{loca}_{top}(\mathsf{loca}_x(-_0))$$

At this point out will have all representatives, in is emptied to indicate termination. (The reader is encouraged to try out the rules on the example location model. It is easiest doing it using the graphical bigraph notation.)

C Rigid control-sortings and RPOs

For a bigraph b (sorted or otherwise), we write b^* for the function that takes each place (site or root) or node of b to its uniquely determined root. In this appendix we will generally omit writing down the link-graph part of interfaces when we do not need them.

Definition C.1 (Rigid control-sorting). *Let $\mathcal{K}$ be a set of controls. A sorting $\mathcal{S} = (\mathcal{K}, \Theta, \Phi)$ is a* rigid control-sorting *if $\Theta \subseteq \mathcal{P}(\mathcal{K})$ and there exists a predicate ϕ, such that*

$$\Phi\Big((m, s_m) \xrightarrow{f} (n, s_n)\Big) \quad \textit{iff} \quad \begin{cases} (i) & s_m(i) = s_n(f^*(i)) & \textit{for } i < m, \\ (ii) & \phi(ctrl_f(v), s_n(f^*(v))) & \textit{for } v \textit{ node in } f. \end{cases}$$

In the sequel, we assume a fixed set of controls $\mathcal{K}$, rigid control-sorting $\mathcal{S} = (\mathcal{K}, \Theta, \Phi)$, a sorted signature $\Sigma^{\mathcal{S}}$ and a corresponding unsorted signature $\mathcal{U}(\Sigma^{\mathcal{S}}) = \Sigma$; following [Jen06], we write $\mathfrak{Big}(\Sigma)$ for the precategory of concrete bigraphs over Σ and $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ for the corresponding precategory of sorted concrete bigraphs, and we write $\mathcal{U}$ for the forgetful functor from $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ to $\mathfrak{Big}(\Sigma)$; recall that this functor is faithful.

In Theorem C.1 we state that $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ has RPOs; it follows that the standard bisimulation on $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ is a congruence. To establish Theorem C.1, we will need some lemmas to make precise just how closely $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ mimics $\mathfrak{Big}(\Sigma)$.

Lemma C.1. *If $\mathcal{U}(a) = p \circ q$, then there exists unique b, c s.t. $\mathcal{U}(b) = p$, $\mathcal{U}(c) = q$ and $a = b \circ c$.*

Proof. For existence, suppose $a : (m, s_m) \longrightarrow (n, s_n)$ and $cod(q) = dom(p) = l$. Define

$$s_l(i) \stackrel{\text{def}}{=} s_n(p^*(i)). \tag{3.11}$$

We claim that $c = (m, s_m) \xrightarrow{q} (l, s_l)$ and $b = (l, s_l) \xrightarrow{p} (n, s_n)$ are well-sorted. Consider c. Condition (i) of Definition C.1 is satisfied by (3.11), Condition (ii) is satisfied because the nodes of c is a subset of the nodes of a. Now consider b. For $i < n$, we find

$$s_m(i) = s_n(a^*(i)) = s_n(p^*(q^*(i))) = s_l(q^*(i)),$$

satisfying Condition (i). Next, for v a node of q, we find

$$\phi(ctrl_q(v), s_l(q^*(v))) = \phi(ctrl_a(v), s_n(p^*(q^*(v)))) = \phi(ctrl_a(v), s_n(a^*(v))).$$

But $\phi(ctrl_a(v), s_n(a^*(v)))$ is satisfied by well-sortedness of a; thus Condition (ii) is satisfied.

For uniqueness, it is sufficient to prove that s_l is the only sorting making b and c well-sorted. Suppose s'_l is an alternate such sorting. If there is $i < l$ s.t. $s'_l(i) \neq s_l(i) = s_n(p^*(i))$, then $(l, s'_l) \xrightarrow{p} (n, s_n)$ is not well-sorted: contradiction. Thus $s'_l = s_l$. □

Lemma C.2. *If a, b is a cospan and $\mathcal{U}(a) = \mathcal{U}(b)$, then $a = b$.*

Proof. Because $\mathcal{U}(a) = \mathcal{U}(b)$, a and b must have the same inner width, m:

$$(m, s_m) \xrightarrow{\quad a \quad} (n, s_n) \xleftarrow{\quad b \quad} (m, s'_m).$$

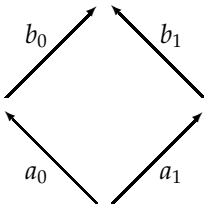
Suppose for a contradiction that there is $i < m$ s.t. $s'_m(i) \neq s_m(i)$. Then

$$s_n(a^*(i)) = s_m(i) \neq s'_m(i) = s_n(b^*(i)),$$

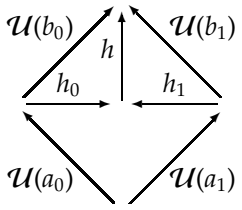
but that cannot be, because $a^*(i) = b^*(i)$ follows from $\mathcal{U}(a) = \mathcal{U}(b)$: contradiction. □

Theorem C.1. $\mathfrak{B}\textsc{ig}(\Sigma^{\mathcal{S}})$ *has RPOs.*

Proof. Consider the square (i) below.

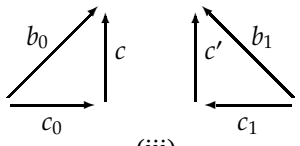


(i)



(ii)

Apply $\mathcal{U}$ to get a similar square in $\mathfrak{B}\textsc{ig}(\Sigma)$, and erect an RPO there, altogether obtaining the diagram (ii). By Lemma C.1, there are c_0 and c factoring b_0 s.t. $\mathcal{U}(c_0) = h_0$ and $\mathcal{U}(c) = h$; symmetrically, there are also c_1 and c' factoring b_1 s.t. $\mathcal{U}(c_1) = h_1$ and $\mathcal{U}(c') = h$. (See diagram (iii) below.)



(iii)

But b_0, b_1 is a cospan, so also c, c' is a cospan; thus $c = c'$ by Lemma C.2, and we have a candidate RPO c_0, c_1, c.

Suppose d_0, d_1, d is an alternate candidate RPO. We must find unique e s.t. $c = d \circ e$. Because $\mathcal{U}(c_0), \mathcal{U}(c_1), \mathcal{U}(c)$ is an RPO, we find unique p s.t. $\mathcal{U}(c) = \mathcal{U}(d) \circ p$. By Lemma C.1, there are unique d', e s.t. $\mathcal{U}(d') = \mathcal{U}(d), \mathcal{U}(e) = p$ and $c = d' \circ e$. But then d, d' is cospan, so by Lemma C.2, $d = d'$. Thus, we have found e s.t. $c = d \circ e$. For uniqueness, suppose there is e' with $c = d \circ e'$. Then

$$h = \mathcal{U}(c) = \mathcal{U}(d) \circ \mathcal{U}(e') = \mathcal{U}(d) \circ p$$

but then $\mathcal{U}(e') = p = \mathcal{U}(e)$ by uniqueness of p; but e', e is also a cospan, so by Lemma C.2, $e = e'$. □

Corollary C.1. *Bisimulation on the standard transition-system of* $\mathfrak{B}\textbf{IG}(\Sigma^{\mathcal{S}})$ *is a congruence.*

Proof. By [Jen06, Theorem 3.16], possession of RPOs is a sufficient prerequisite for the desiderata. □

We can now prove that the sorting of Definition 3.4.5 gives a congruential bisimulation.

Theorem C.2. *Let* $\mathcal{S}$ *be the sorting given in Definition 3.4.5. Then the bisimulation over the standard transitions of* $\mathfrak{B}\textbf{IG}(\Sigma^{\mathcal{S}})$ *is a congruence.*

Proof. By Corollary C.1, it is sufficient to show that $\mathcal{S}$ is a rigid control sorting. Take $\phi(k, K) = k \in K$. Clearly, $\Phi\Big((m, s_m) \xrightarrow{f} (n, s_n)\Big)$ is equivalent to $i < m \implies s_m(i) = s_n(f^*(i))$ and $v \in f \implies \phi(ctrl_f(v), s_n(f^*(v)))$. □

Chapter 4

Encoding MiniML with References in Bigraphs

4.1 Purpose

In this chapter we encode a MiniML-like calculus with references, Ξ, in bigraphs. The motivation is two-fold: (1) We would like to be able to express some parts of our bigraphical location model (see chapter 5) in Ξ utilising the Plato-graphical ability to combine several languages. (2) It is an interesting study in itself to investigate how one may model references in bigraphs, because to the best of our knowledge no previous encodings of calculi with side effects have been studied in bigraphs.

Recall that we found it useful to be able to express some parts of our system in a high-level language, chapter 3. We claim that Ξ is such a language. We need references for the location model part **L** and the agent part **A** of our location model in chapter 5. For now, we ask the reader to trust us when we say that references are needed for encapsulating the state of the location model wrt. the agent and the sensor.

We investigate the use of closed links in *local bigraphs* as defined in [Mil04c, Mil04d]. This chapter aims to

- further investigate a question of chapter 3, namely which high-level languages that can be encoded on bigraphs for use in Plato-garphical models,
- communicate a few humble insights about the behaviour of closed links (edges) to readers with some knowledge of bigraphs,
- to experiment with and illustrate some non-trivial *matches*,
- and to show how references can be encoded in (local) bigraphs using closed links.

First, we find that closed links can not interfere with outer names (open links) or with each other, and that they behave as they do in *binding* bigraphs when they are replicated as part of a parameter. Secondly, we find that a calculus with references can be encoded in (local) bigraphs using closed links. There are some subtleties associated with the encoding and the resulting reaction relation, which we address as we go along.

4.2 Non-interference of closed links

Recall that the *arity* of a control $\mathsf{K} : b \to f$ is a pair of finite ordinals, b is the *binding arity* and f the *free arity*. Consider the following binding signature Σ with a parametric reaction rule.

Control	Activity	Arity	Comment
loca	active	$0 \to 1$	Nested location (e.g. a room)
devi	atomic	$0 \to 1$	Mobile device

$$\mathsf{loca}_g(-_0) \longrightarrow\!\!\!\!\!\rightarrow\ /x\,.\,\mathsf{loca}_g(\mathsf{devi}_x \mid -_0) \tag{4.1}$$

NOTATION. We introduce the *operator* $/x\,.\,B$ as a generalisation of $/x \circ B$, in the sense that $/x\,.\,B$ works for all widths of B, and it marks where identities are implicit in a composition. It binds less than |, ||, and /. | binds tighter than || which binds tighter than /.

By Def. 3.2 (parametric reaction rule) of [Mil04c] and the additional clarification in section 3 of [Mil04d], our parametric reaction rule (4.1) is of the form

$$(R : I \to K, R' : I' \to K, \eta, \vec{\iota})$$

where $I = \vec{X}$ and $I' = \vec{X'}$ are partitions (i.e. disjoint) with widths m and m', and $\eta : m' \to m$ is a map of ordinals. The fourth component is a vector of bijections $\vec{\iota_j} : X_{\eta(j)} \to X'_j$ for each $j \in m'$.

A parametric rule generates ground rules of the form

$$((R \oplus \omega) \circ a, (R' \oplus \omega') \circ a')$$

where $I \oplus H$, $I' \oplus H'$ and $K \oplus L$ are interface extensions with $H' = \overline{\eta}(H)$[1]. Let $\omega : H \to L$ and $\omega' : H' \to L$ be wirings that agree on the names of H' and have the same support, i.e. $|\omega| = |\omega'|$. Then for any $a : I \oplus H$, complete the ground rule by defining $a' = \overline{\eta}_{\vec{\iota}}(a) : I' \oplus H'$.

NOTATION. We follow the short-hand notation of [Mil04c] and write a local interface as a vector of names. We omit the parentheses if the vector is of size one, and

[1] $\overline{\eta}$ is the *instantiation* map.

the curly brackets if the set is a singleton. We omit the empty inner face ϵ of ground bigraphs.

The components of (4.1) are as follows.

$$\begin{aligned} R &= \mathsf{loca}_g(-_0) : \emptyset \to g \\ R' &= /x \,.\, \mathsf{loca}_g(\mathsf{devi}_x \mid -_0) : \emptyset \to g \\ \eta &= \mathsf{Id}_1 \\ \vec{\iota} &= (\mathsf{Id}_\emptyset) \end{aligned}$$

where Id_1 is the identity function on the finite ordinal $1 = \{0\}$, and $\vec{\iota}$ is a vector of isomorphisms relating located names in the parameters (none in this case). We see that $x \neq g$ since otherwise $cod(R) \neq cod(R')$. Thus, we could just as well have chosen $R' = \mathsf{loca}_g(/x \,.\, \mathsf{devi}_x \mid -_0)$, since these two terms denote the same bigraph, which is apparent graphically, and justified by Prop. 2.7 (open decomposition) of [Mil04c]. In the term language, however, one must be careful when choosing names; if $x = g$, then $\mathsf{loca}_g(/x \,.\, \mathsf{devi}_x \mid -_0) \neq /x \,.\, \mathsf{loca}_g(\mathsf{devi}_x \mid -_0) = (\mathsf{id}_g \oplus /x) \circ \mathsf{loca}_g(\mathsf{devi}_x \mid -_0)$ because $\mathsf{id}_g \oplus /x$ is not defined when $g = x$. We see that when writing rules a closed link can not "capture" outer names by accident.

Now consider, informally, the following situation: What happens if we apply (4.1) twice consecutively? If we think of a device's outer name as its identifier, can we accidently reuse (the name of) a closed link and thereby identify two different devices? I.e. can we formally have the following sequence of reactions: $\mathsf{loca}_p() \longrightarrow /f \,.\, \mathsf{loca}_p(\mathsf{devi}_f) \longrightarrow /f \,.\, \mathsf{loca}_p(\mathsf{devi}_f \mid \mathsf{devi}_f)$? The answer is no; the second reaction is not allowed. Let us convince ourselves by looking at an example.

Consider a bigraph $B_1 = \mathsf{loca}_p()$, then $B_1 \longrightarrow B_2$ if and only if there exist $C, \omega, a, \omega', a'$ such that

$$\begin{aligned} B_1 &= C \circ (R \oplus \omega) \circ a \\ B_2 &= C \circ (R' \oplus \omega') \circ a' \end{aligned}$$

where ω is a wiring and parameter a is discrete (for $\emptyset$ and g). Clearly, we must have

$$\begin{aligned} a &= 1 : \emptyset \\ \omega &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ C &= p/g : g \to p \\ a' &= a \\ \omega' &= \omega \end{aligned}$$

where $B_2 = /f \,.\, \mathsf{loca}_p(\mathsf{devi}_f) = \mathsf{loca}_p(/f \,.\, \mathsf{devi}_f)$ for any $f \neq p$ (forced by the rule). Clearly, the choice of the name f is insignificant, because it does not appear in the outer (or inner) face of B_2 when closed, and since edges (closed links) do not have identity in abstract bigraphs. Thus, $/f \,.\, \mathsf{loca}_p(\mathsf{devi}_f) = /k \,.\, \mathsf{loca}_p(\mathsf{devi}_k)$ for any $f, k \neq p$.

We wish to apply (4.1) to B_2. B_2 can be matched by (4.1) in two ways depending on whether the closure of f is done in the wirings ω, ω' or in the context C. If we close f in the context then the wirings ω, ω' will be *placings*, and Prop. 3.5 (placings suffice) of [Mil04c] states that it suffices to consider such ground rules. To accentuate this we write π, π' for the wirings (placings), and we have

$$\begin{aligned} a &= \mathsf{devi}_f : f \\ \pi &= \mathsf{id}_f : f \to f \\ C &= p/g \oplus /f : \{g, f\} \to p \\ a' &= a \\ \pi' &= \pi \end{aligned}$$

where $B_3 = /h \,.\, /f \,.\, \mathsf{loca}_p(\mathsf{devi}_h \mid \mathsf{devi}_f)$ for arbitrary $f, h \neq p$. Graphically, it is clear that the order of closures is insignificant so this also holds in the term language. Could we have chosen $f = h$? No, because that would render C undefined. We conclude that closed links can not interfere with one another, which is apparent in the graphical representation.

Now, let us consider replication of parameters involving closed links in local bigraphs. To illustrate, we consider the "replication rule" of [JM04] (p. 78):

$$\begin{aligned} R &= \mathsf{rep}(-_0) : \emptyset \to \emptyset \\ R' &= -_0 \mid \mathsf{rep}(-_1) : (\emptyset, \emptyset) \to \emptyset \\ \eta &= \{0, 1 \mapsto 0\} \\ \vec{\iota} &= (\mathsf{Id}_\emptyset, \mathsf{Id}_\emptyset) \end{aligned}$$

Notice how this rule does not say anything about linking. Applying this rule to the bigraph $B = /l \,.\, \mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l)$ we obtain $B' = /l \,.\, (\mathsf{u}_l \mid \mathsf{v}_l) \mid \mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l)$ because

$$\begin{aligned} a &= \mathsf{u}_x \mid \mathsf{v}_y : \{x, y\} \\ \pi &= \mathsf{id}_{\{x,y\}} : \{x, y\} \to \{x, y\} \\ C &= /l \,.\, (l/x \mid l/y) : \{x, y\} \to \emptyset \\ a' &= (\mathsf{u}_x \mid \mathsf{v}_y) \parallel (\mathsf{u}_x \mid \mathsf{v}_y) \\ \pi' &= \mathsf{id}_{\{x,y\}} \mid \mathsf{id}_{\{x,y\}} : (\{x, y\}, \{x, y\}) \to \{x, y\} \end{aligned}$$

The closure resides in C so the global outer names of the two replicas are identified (must point to the same edge). If one wishes each replica to have its own private link, one should use binding ports or replicate using, e.g., the following rule, which is a one-time replication rule for simplicity.

$$\begin{aligned} R &= /x \,.\, \mathsf{rep}(-_0\langle x \rangle) : \{x\} \to \emptyset \\ R' &= /x \,.\, -_0\langle x \rangle \mid /y \,.\, -_1\langle y \rangle : (x, y) \to \emptyset \\ \eta &= \{0, 1 \mapsto 0\} \\ \vec{\iota} &= (x \mapsto x, x \mapsto y) \end{aligned}$$

With this rule we have $B = /l\,.\,\mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l) \longrightarrow B' = (/l\,.\,\mathsf{u}_l \mid \mathsf{v}_l) \mid (/l\,.\,\mathsf{u}_l \mid \mathsf{v}_l)$. We show the components of the match to illustrate the "trick" of closing the links in the rule so that the context can not identify outer names before closing them. The intuition is that the context C has no handle on the links and thus can not manipulate them; x is closed in R sp u and v in a can not have different outer names because C can not reunite them.

$$\begin{aligned} a &= \mathsf{u}_x \mid \mathsf{v}_x : x \\ \omega &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ C &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ a' &= \mathsf{u}_x \mid \mathsf{v}_x \parallel \mathsf{u}_i \mid \mathsf{v}_j : (x, y) \\ \omega' &= \mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset : (\emptyset, \emptyset) \to \emptyset \end{aligned}$$

This concludes our treatment of closed link non-interference.

4.3 Encoding references via closed links

Consider the following MiniML-like call-by-value calculus Ξ with pairs and projections, references, datatype constructors and deconstructors, fixed-points, and natural numbers. We need some notation.

NOTATION. Denote the dereferencing operation by ! (bang). n ranges over the set $\mathcal{N}$ of natural numbers (including zero). x, f range over an infinite set $\mathcal{V}$ of variable names with members x, y, z and so forth. l ranges over a set $\mathcal{L}$ of *reference cells*. D ranges over an infinite set $\mathcal{D}$ of contructor names. C ranges over an infinite set $\mathcal{C}$ of contructor names with members C0, C1, C2 and so forth. The sets $\mathcal{V}$, $\mathcal{L}$, $\mathcal{D}$, and $\mathcal{C}$ are pairwise disjoint. We use the shorthand notation 'case e of $C_i\ x_i \Rightarrow e_i\ ^{i=0..n}$' for 'case e of $C_0\ x_0 \Rightarrow e_0 \mid C_1\ x_1 \Rightarrow e_1 \mid \ldots \mid C_n\ x_n \Rightarrow e_n$'. Likewise for data-type declarations.

Here is the calculus Ξ in form of a BNF grammar with programs p, terms e, values v, and evaluation contexts E.

$$\begin{aligned} p &::= \text{datatype D} = \text{C}_i \text{ of } t_i\ ^{i=0..n} ;\ p \ [\!]\ e \\ e &::= x \ [\!]\ e_1\ e_2 \ [\!]\ (e_1, e_2) \ [\!]\ \text{fst}\ e \ [\!]\ \text{snd}\ e \ [\!]\ \text{let}\ x = e_1\ \text{in}\ e_2 \ [\!] \\ &\quad\ \text{ref}\ e \ [\!]\ !e \ [\!]\ e_1 := e_2 \ [\!]\ \text{C}\ e \ [\!]\ \text{case}\ e\ \text{of}\ \text{C}_i\ x_i \Rightarrow e_i\ ^{i=0..n} \ [\!]\ v \\ v &::= \lambda x.e \ [\!]\ \text{fix}\ f(x) = e \ [\!]\ (v_1, v_2) \ [\!]\ \text{unit} \ [\!]\ l \ [\!]\ \text{C}\ v \ [\!]\ n \\ E &::= [\,] \ [\!]\ (E, e) \ [\!]\ (v, E) \ [\!]\ \text{fst}\ E \ [\!]\ \text{snd}\ E \ [\!]\ \text{let}\ x = E\ \text{in}\ e \ [\!]\ \text{let}\ x = v\ \text{in}\ E \ [\!] \\ &\quad\ E\ e \ [\!]\ v\ E \ [\!]\ \text{ref}\ E \ [\!]\ !E \ [\!]\ E := e \ [\!]\ v := E \ [\!]\ \text{C}\ E \ [\!] \\ &\quad\ \text{case}\ E\ \text{of}\ \text{C}_i\ x_i \Rightarrow e_i\ ^{i=0..n} \end{aligned}$$

A program is a possibly empty sequence of data-type declarations followed by an expression. When writing concrete programs we will typically omit ';'. Concrete

cell constants (ranged over by l) only arise in terms that are the intermediate results of evaluation; they are not in the language in which programmers write. We have chosen to explicitly include several constructs in the calculus that could have been encoded instead. This is done to avoid cluttering the presentation and use of the calculus with encodings. Should we, however, wish to formally prove properties about this calculus then encodings would be preferred to limit the number of cases in inductive analyses.

We introduce a *store* to keep track of *store cell* values, and define dynamics via a single-step evaluation relation $\longrightarrow$ on *configurations*. We use the following notational conventions.

NOTATION. σ ranges over stores, i.e. partial functions from locations to values, $(\sigma, l \mapsto v)$ denotes binding, and $\sigma[l \mapsto v]$ denotes updating.

We call the following rules *basic*.

$$
\begin{array}{rcll}
\langle \text{fst}\ (v_1, v_2), \sigma\rangle & \longrightarrow & \langle v_1, \sigma\rangle & \\
\langle \text{snd}\ (v_1, v_2), \sigma\rangle & \longrightarrow & \langle v_2, \sigma\rangle & \\
\langle \text{let}\ x = v\ \text{in}\ e, \sigma\rangle & \longrightarrow & \langle e\{v/x\}, \sigma\rangle & \\
\langle \text{ref}\ v, \sigma\rangle & \longrightarrow & \langle l, (\sigma, l \mapsto v)\rangle \quad , & l \in \mathcal{L}\ \textit{fresh} \\
\langle !l, \sigma[l \mapsto v]\rangle & \longrightarrow & \langle v, \sigma[l \mapsto v]\rangle & \\
\langle l := v', \sigma[l \mapsto v]\rangle & \longrightarrow & \langle \text{unit}, \sigma[l \mapsto v']\rangle & \\
\langle (\lambda x.e)\ v, \sigma\rangle & \longrightarrow & \langle e\{v/x\}, \sigma\rangle & \\
\langle (\text{fix}\ f(x) = e)\ v, \sigma\rangle & \longrightarrow & \langle e\{v/x, (\text{fix}\ f(x) = e)/f\}, \sigma\rangle & \\
\langle \text{case}\ C_j\ v\ \text{of}\ C_i\ x_i \Rightarrow e_i\ ^{i=0..n}, \sigma\rangle & \longrightarrow & \langle e_j\{v/x_j\}, \sigma\rangle \quad , & if\ j \in \{0, \ldots, n\}
\end{array}
$$

Notice that stores are *not* terms. We briefly explain the last rule. It allows evaluation of a case construct provided that the expression to be matched is a value, and that it matches one of the contructors declared. The result of the evaluation is a substitution of v for x_j in the j^{th} branch of the case contruct. There is only one variable on the left-hand side in each branch, which means that if one wishes to match a constructor with a value that is, e.g., a pair then the variable x_j has to be manually deconstructed (in this case projected) on the right-hand side of the matching branch. We will see an example of this in chapter 5. We close evaluation under contexts:

Definition 4.3.1. *If $\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$ then there exists a unique E such that $e = E[r]$, $\langle r, \sigma\rangle \longrightarrow \langle r', \sigma'\rangle$ is a reduction by one of the* basic *rules, and $e' = E[r']$.*

We encode Ξ in local bigraphs. The signature is shown in figure 4.1. For now, let $\mathsf{i0}, \mathsf{i1}, \mathsf{i2}, \ldots$ represent natural numbers. These are used instead of a more cumbersome bigraphical representation, which will be presented in subsection 4.3.1, of natural numbers using only constants *zero* and *successor*. Binding ports are used to delimit variable scope (see e.g. letb, lam, and casee). Some controls are passive to prevent evaluation "under them". The last six controls are not images of Ξ-terms,

Control	Activity	Arity	Comment
var	atomic	$0 \to 1$	Variable
app	active	$0 \to 0$	Application
appl	active	$0 \to 0$	Left part of application
appr	active	$0 \to 0$	Right part of application
pair	active	$0 \to 0$	Pair
pairl	active	$0 \to 0$	Left component of pair
pairr	active	$0 \to 0$	Right component of pair
fst	active	$0 \to 0$	Left projection on pair
snd	active	$0 \to 0$	Right projection on pair
let	active	$0 \to 0$	Let construction
letd	active	$0 \to 0$	Declaration of 'let'
letb	passive	$1 \to 0$	Body of 'let'
ref	active	$0 \to 0$	Reference request
deref	active	$0 \to 0$	Dereference
assign	active	$0 \to 0$	Assignment
acell	active	$0 \to 0$	Cell component of assignment
aval	active	$0 \to 0$	Value component of assignment
case	active	$0 \to 0$	Case construct
casel	active	$0 \to 0$	Constructor to be matched of 'case'
casee	passive	$1 \to 0$	Case branch of 'case'
Cn	active	$0 \to 0$	A control for each member of C
val	passive	$0 \to 0$	Value
lam	passive	$1 \to 0$	Lambda abstraction
fix	passive	$2 \to 0$	Fixed-point construction
unit	atomic	$0 \to 0$	Unit, for side effects
i0, i1, i2 ...	atomic	$0 \to 0$	Natural numbers
cell	atomic	$0 \to 1$	Generated reference cell (term)
cell'	passive	$0 \to 1$	Generated store cell
store	active	$0 \to 0$	Store
exp	passive	$0 \to 0$	Delay evaluation
sub	active	$1 \to 0$	Term to undergo substitution
def	active	$0 \to 1$	Term to be inserted

Figure 4.1: Signature Σ_{Ξ}.

but are introduced to implement a store, call-by-value evaluation, and substitution. cell denotes a reference cell generated by evaluation. cell′ denotes a store cell, i.e. it resides in store, holding the value of a reference cell, which it refers to by a closed link. As will become apparent when translating Ξ-terms into local bigraphs, exp is used to delay evaluation certain places in terms to implement the call-by-value semantics. sub and def are used to perform explicit substitution, sub holds the term to undergo substitution for some x and def holds the value to be inserted instead of x. The purposes of the other controls should be clear.

We provide a formal translation of Ξ into local bigraphs assuming well-formed Ξ-programs. The semantic function $(\!|\cdot|\!)$ translates programs p, and the semantic function $[\![\cdot]\!]$ translates expressions e. The idea is to preserve the call-by-value semantics by inserting passive exp controls to hinder premature evaluation. Binding ports are used to limit the scope of variables, like lambda abstractions in the λ-calculus.

NOTATION. We write $f \circ g$ as fg, and composition binds tighter than prime product (on the right-hand sides).

$$
\begin{aligned}
(\!|\text{datatype D} = \text{C}_i \text{ of } t_i\ ^{i=0..n}\ ;\ p|\!)_X &= (\!|p|\!)_X \\
(\!|\text{datatype D} = \text{C}_i \text{ of } t_i\ ^{i=0..n}\ ;\ e|\!)_X &= [\![e]\!]_X \\
[\![x]\!]_{X \uplus \{x\}} &= X \oplus \mathsf{var}_x \\
[\![(e_1, e_2)]\!]_X &= (\mathsf{pair} \oplus \mathsf{id}_X) \\
&\quad \Big((\mathsf{pairl} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
&\quad\ (\mathsf{pairr} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{fst } e]\!]_X &= (\mathsf{fst} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{snd } e]\!]_X &= (\mathsf{snd} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{let } x = e_1 \text{ in } e_2]\!]_X &= (\mathsf{let} \oplus \mathsf{id}_X) \\
&\quad \Big((\mathsf{letd} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
&\quad\ (\mathsf{letb}_{(x)} \oplus \mathsf{id}_X)[\![e_2]\!]_{X \uplus \{x\}}\Big) \\
[\![\lambda x.e]\!]_X &= (\mathsf{val} \oplus \mathsf{id}_X)(\mathsf{lam}_{(x)} \oplus \mathsf{id}_X)[\![e]\!]_{X \uplus \{x\}} \\
[\![e_1\ e_2]\!]_X &= (\mathsf{app} \oplus \mathsf{id}_X) \\
&\quad \Big((\mathsf{appl} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
&\quad\ (\mathsf{appr} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{ref } e]\!]_X &= (\mathsf{ref} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![!e]\!]_X &= (\mathsf{deref} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![e_1 := e_2]\!]_X &= (\mathsf{assign} \oplus \mathsf{id}_X) \\
&\quad \Big((\mathsf{acell} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
&\quad\ (\mathsf{aval} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{C } e]\!]_X &= (\mathsf{C} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{case } e \text{ of } C_i\ x_i \Rightarrow e_i\ ^{i=0..n}]\!]_X &= (\mathsf{case} \oplus \mathsf{id}_X) \\
&\quad \Big((\mathsf{casel} \oplus \mathsf{id}_X)[\![e]\!]_X \mid \\
&\quad\ (\mathsf{casee}_{(x_1)} \oplus \mathsf{id}_X)(\mathsf{C1} \oplus \mathsf{id}_X) \\
&\quad\quad (\mathsf{exp} \oplus \mathsf{id}_X)[\![e_1]\!]_{X \uplus x_1} \\
&\quad\ \vdots \\
&\quad\ (\mathsf{casee}_{(x_n)} \oplus \mathsf{id}_X)(\mathsf{C}n \oplus \mathsf{id}_X) \\
&\quad\quad (\mathsf{exp} \oplus \mathsf{id}_X)[\![e_n]\!]_{X \uplus x_n}\Big) \\
[\![\text{unit}]\!]_X &= (\mathsf{val} \mid X)\ \mathsf{unit} \\
[\![n]\!]_X &= (\mathsf{val} \mid X)\ \mathsf{i}n \quad , \quad \forall n \in \mathcal{N}
\end{aligned}
$$

The subscript set X on the translation *includes* the names of all free variables (*fv*) in

e. Thus, each term e has many bigraph images. All the images of $[\![\cdot]\!]_X$ are ground so they all have the empty inner face ϵ, and the subscript set as outer face. We require λ-terms to be α-converted in such a way that the binding variables are all different. α-convertible Ξ-terms have equal images, e.g. $[\![\lambda y.y]\!]_\emptyset = [\![\lambda z.z]\!]_\emptyset$. Notice how we do *not* translate configurations, merely terms. This is because the store is implicit in the Ξ-world, but we need to model it explicitly in bigraphs. The store is assumed to be empty at the start of evaluation.

The parametric bigraphical reaction rules corresponding to the dynamic single-step semantics of Ξ are presented in figure 4.2. There is an informal dynamic correspondence between $\longrightarrow$ and $\longrightarrow\!\!\!\!\!\rightarrow$, namely that $\longrightarrow$ can be mimicked by one or more uses of $\longrightarrow\!\!\!\!\!\rightarrow$. During evaluation some variables and cells may disappear and new cells may be created. We demand that $fv(e) \subseteq X$, which is an invariant. The first rule we wish to emphasise is rule (4.6). Evaluation of a 'let' expression results in a substitution. We emphasise the fact that $-_0$ in the redex can have x in its outer face and maintain it in the reactum as explained earlier in this chapter. (4.7) needs to be a wide rule because we wish to have the store at "top level" and not embedded in the program as such. The link is closed outermost immediately upon creation to only grant the freshly generated reference cell cell_l access to that store cell. Thus, we know that l is not in $cod(-_0), cod(-_1)$. Dereferencing and assignment also work as one would expect, (4.8), (4.9) and (4.10), the extra outer name l being necessary to maintain the outer face. This prime product is always defined because we know that $-_0$ is a value and thus has outer width 1 (like $l/$). l is closed, but this is *not* explicit in rules (4.8)-(4.10) because the context has to be able to identify the name l of different regions. In the second assignment rule (4.10) we extend $\mathsf{val}(\mathsf{unit})$ with the wiring $\{l\} : \epsilon \to \{l\}$ to maintain l in the *outer* face of the first region of the reactum. Applying the fixed-point construction to a value results in a nested substitution, as one would expect. We have a rule pair (4.13) and (4.14) for each declared constructor in the source program. In rule (4.15) we exploit local bigraphs by using a wide reaction rule in connection with binding to substitute "at a distance" one occurrence at a time[2]. Furthermore, notice how we have written two sites as $-_0$ in R' implicitly defining $\vec{\iota} = \{0, 1 \mapsto 0\}$. (4.16) terminates the substitution when done; it can not be used before we are actually done substituting because $-_0$ can not contain x in its outer face since in that case $cod(R) \neq cod(R')$. The rest of the rules should be clear.

4.3.1 Encoding of natural numbers

Here we present an encoding of natural numbers in bigraphs using a Peano-like representation with zero and successor. We also encode rules implementing simple operations like equality on these using an encoding of boolean constants True and False. There should be no surprises in figure 4.3. The rules presented in figure 4.4 should be straightforward, but do notice that we use non-negative subtraction on integers (so really we restrict ourselves to natural numbers) in rule (4.28) and that multiplication is computed according to the formula $(m+1)*(n+1) = (n+1)+m*$

[2]This does not work in *binding* bigraphs because here a name can not reside in multiple locations.

$$\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{exp}(-_1))) \longrightarrow \mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(-_1)) \quad (4.2)$$

$$\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{val}(-_1))) \longrightarrow \mathsf{val}(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{val}(-_1)))) \quad (4.3)$$

$$\mathsf{fst}(\mathsf{val}(\mathsf{pair}(\mathsf{pairl}(-_0) \mid \mathsf{pairr}(-_1)))) \longrightarrow -_0 \quad (4.4)$$

$$\mathsf{snd}(\mathsf{val}(\mathsf{pair}(\mathsf{pairl}(-_0) \mid \mathsf{pairr}(-_1)))) \longrightarrow -_1 \quad (4.5)$$

$$\mathsf{let}(\mathsf{letd}(\mathsf{val}(-_0)) \mid \mathsf{letb}_{(x)}(-_1\langle x\rangle)) \longrightarrow \mathsf{sub}_{(x)}(-_1\langle x\rangle \mid \mathsf{def}_x(\mathsf{val}(-_0))) \quad (4.6)$$

$$\mathsf{ref}(\mathsf{val}(-_0)) \parallel \mathsf{store}(-_1) \longrightarrow /l\,.\,\mathsf{val}(\mathsf{cell}_l) \parallel \mathsf{store}(\mathsf{cell}'_l(\mathsf{val}(-_0)) \mid -_1) \quad (4.7)$$

$$\mathsf{deref}(\mathsf{val}(\mathsf{cell}_l)) \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_1) \longrightarrow -_0 \mid l/ \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_1) \quad (4.8)$$

$$\mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(\mathsf{exp}(-_0))) \longrightarrow \mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(-_0)) \quad (4.9)$$

$$\mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(\mathsf{val}(-_0))) \parallel \mathsf{store}(\mathsf{cell}'_l(-_1) \mid -_2) \longrightarrow \mathsf{val}(\mathsf{unit}) \oplus \{l\} \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_2) \quad (4.10)$$

$$\mathsf{app}(\mathsf{appl}(\mathsf{val}(\mathsf{lam}_{(x)}(-_0\langle x\rangle))) \mid \mathsf{appr}(\mathsf{val}(-_1))) \longrightarrow \mathsf{sub}_{(x)}(-_0\langle x\rangle \mid \mathsf{def}_{(x)}(-_1)) \quad (4.11)$$

$$\mathsf{app}(\mathsf{appl}(\mathsf{val}(\mathsf{fix}_{(f,x)}(-_0\langle f, x\rangle))) \mid \mathsf{appr}(\mathsf{val}(-_1))) \longrightarrow \mathsf{sub}_{(f)}(\mathsf{sub}_{(x)}(-_0\langle x\rangle \mid \mathsf{def}_x(\mathsf{val}(-_1))) \mid \mathsf{def}_f(\mathsf{val}(\mathsf{fix}_{(f,x)}(-_0\langle f, x\rangle)))) \quad (4.12)$$

$$\mathsf{C}(\mathsf{val}(-_0)) \longrightarrow \mathsf{val}(\mathsf{C}(\mathsf{val}(-_0))) \quad (4.13)$$

$$\mathsf{case}(\mathsf{casel}(\mathsf{val}(\mathsf{C}(-_0))) \mid (\mathsf{casee}_{(x)}(\mathsf{C}(\mathsf{exp}(-_1\langle x\rangle))) \mid -_2)) \longrightarrow \mathsf{sub}_{(x)}(-_1\langle x\rangle \mid \mathsf{def}_x(-_0)) \quad (4.14)$$

$$\mathsf{var}_x \parallel \mathsf{def}_x(\mathsf{val}(-_0)) \longrightarrow \mathsf{val}(-_0) \mid \{x\} \parallel \mathsf{def}_x(\mathsf{val}(-_0)) \quad (4.15)$$

$$\mathsf{sub}_{(x)}(-_0 \mid \mathsf{def}_x(-_1)) \longrightarrow -_0 \quad (4.16)$$

Figure 4.2: Reaction rules for Ξ.

$(n + 1)$ in rule (4.33). As mentioned in chapter 4 we introduce a shorthand notation for natural numbers; i0 means z, i1 means s z, i2 means s (s z), and so forth.

In an implementation of a bigraphical rewrite engine, it is probably too inefficient to work with bigraphical integers (and booleans) so one would perhaps choose to work with the integers (and booleans) of the implementation language instead. Using i−controls is a more human-readable representation.

Control	Activity	Arity	Comment
true	atomic	$0 \to 0$	True
false	atomic	$0 \to 0$	False
z	atomic	$0 \to 0$	Zero (i0)
s	active	$0 \to 0$	Successor
argl	active	$0 \to 0$	Left argument
argr	active	$0 \to 0$	Right argument
eqi	active	$0 \to 0$	Equality operator
lti	active	$0 \to 0$	Less than operator
addi	active	$0 \to 0$	Addition operator
subi	active	$0 \to 0$	Subtraction operator
muli	active	$0 \to 0$	Multiplication operator

Figure 4.3: Signature for natual numbers encoding in local bigraphs.

4.3.2 An example exploring references

In this chapter we wish to explore the encoding of references, and this only uses a fragment of Ξ. We shall return to the other parts of Ξ in chapter 5. Consider the program shown in figure 4.5. It is written in Ξ, and translated into bigraphs. Clearly, the result of evaluating this program should be the value 4 (i4). This program uses *aliasing*, and we intend to see if the encoding behaves correctly. We have presented the bigraph in short-hand notation leaving wirings implicit.

Denote by A_0 the bigraph shown in figure 4.5. A_0 rewrites to $A_{14} = /l\,.\,\mathsf{val(i4)} \mid \mathsf{store(loc}'_l\mathsf{(val(i4)))}$ via the reduction path (4.7) – (4.6) – (4.15) – (4.6) – (4.15) – (4.16) – (4.9) – (4.10) – (4.2) – (4.15) – (4.16) – (4.8) – (4.3) – (4.5). The most interesting steps are the uses of rules (4.7), (4.10), and (4.8), but we also show how rules (4.6), (4.15), and (4.16) are matched because there are some subtleties here. We only show the maps $\eta, \vec{\iota}$ in the matches where they are important and non-trivial. The reader may want to look up these rules when inspecting the matches in what follows.

NOTATION. We use S, T, U to denote certain sub-terms of a bigraph when these sub-terms are not important for the particular match, to increase readability when presenting the matches.

We begin to evaluate the declaration part of the outermost let; A_0 matches (4.7) because we find

$$\begin{aligned} a &= \mathsf{i5} \parallel 1 : (\emptyset, \emptyset) \\ \omega &= \mathsf{id}_2 : (\emptyset, \emptyset) \to (\emptyset, \emptyset) \\ C &= \mathsf{let}\Big(\mathsf{letd}(-_2) \mid \mathsf{letb}_{(z)}(S)\Big) \mid -_3 : (\emptyset, \emptyset) \to \emptyset \\ a &= a' \\ \omega' &= 1 \otimes (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset) \to (\emptyset, \emptyset) \end{aligned}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow \mathsf{true} \tag{4.17}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \longrightarrow \mathsf{eqi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.18}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{s}(-_0))) \longrightarrow \mathsf{false} \tag{4.19}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow \mathsf{false} \tag{4.20}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow \mathsf{false} \tag{4.21}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \longrightarrow \mathsf{lti}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.22}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{s}(-_0))) \longrightarrow \mathsf{true} \tag{4.23}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow \mathsf{false} \tag{4.24}$$

$$\mathsf{addi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \longrightarrow -_0 \tag{4.25}$$

$$\mathsf{addi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow -_0 \tag{4.26}$$

$$\begin{aligned}&\mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1)))\\ &\quad\longrightarrow\quad \mathsf{s}(\mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(-_1)))\end{aligned} \tag{4.27}$$

$$\mathsf{subi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \longrightarrow \mathsf{z} \tag{4.28}$$

$$\mathsf{subi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow -_0 \tag{4.29}$$

$$\mathsf{subi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \longrightarrow \mathsf{subi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.30}$$

$$\mathsf{muli}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \longrightarrow \mathsf{z} \tag{4.31}$$

$$\mathsf{muli}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \longrightarrow \mathsf{z} \tag{4.32}$$

$$\begin{aligned}&\mathsf{muli}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1)))\\ &\quad\longrightarrow\quad \mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_1)) \mid \mathsf{argr}(\mathsf{muli}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{s}(-_1)))))\end{aligned} \tag{4.33}$$

Figure 4.4: Operations on integers in local bigraphs.

where S denotes the content of the outermost letb in A_0. It is easy to convince oneself that $A_0 = C \circ (R \oplus \omega) \circ a$. Consider an A_1 s.t. $A_0 \longrightarrow A_1$:

$$A_1 = /l \,.\, \mathsf{let}\Big(\mathsf{letd}(\mathsf{val}(\mathsf{loc}_l)) \mid \mathsf{letb}_{(z)}(S)\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5})))$$

Notice how in A_1 the closure is done "outermost" whereas in the rule it is done in R'. This situation is essentially the question whether

$$\Big(\mathsf{F}(-_0) \mid \mathsf{G}(-_1)\Big) \circ /l \,.\, \mathsf{H}_l \mid \mathsf{I}_l = /l \,.\, \mathsf{F}(\mathsf{H}_l) \mid \mathsf{G}(\mathsf{I}_l)$$

for some ions $\mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{I}$. Looking at how composition is defined in Def. 8.3 (precate-

```
let z = ref 5
in let y = z in
snd (y:=4,!z)

-------------------------------------------------

let(letd(ref(val(i5))) |
    letb_(z)(
      let(letd(var_z) |
          letb_(y)(
            snd(pair(
                  pairl(assign(
                          aloc(var_y) |
                          aval(exp(i4)))) |
                  pairr(exp(deref(var_z))))))))) |
store()
```

Figure 4.5: A Ξ-program and its translation into bigraphs.

gory of link graphs) of [JM04], we see that case one of the definition of the composite link map applies so indeed the equation holds, and this also applies to our setting. (It also holds for ‖.) Thus, $A_1 = C \circ (R' \oplus \omega') \circ a'$ as required. Intuitively, it may seem weird that one can plug in two link-connected regions (when ‖ is used) into two separate holes and keep the connection intact – especially when one looks at the graphical representation. However, the closed link is a visual deception – formally the link map just maps the outer name l of H and I to an edge, and this relationship is maintained during composition.

We continue by showing how A_1 matches (4.6). We have $\eta = \mathsf{Id}_2$ and $\vec{\iota} = (\mathsf{Id}_\emptyset, \mathsf{Id}_x)$. When matching we can wlog. rename z to x in A_1 because the two terms denote the same bigraph.

$$\begin{aligned} a &= \mathsf{loc}_l \parallel S : (l, x) \\ \omega &= \mathsf{id}_l \mid \mathsf{id}_\emptyset : (l, \emptyset) \to l \\ C &= /l \,.\, \mathsf{id}_l \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val(i5)})) : l \to \emptyset \\ a' &= a \\ \omega' &= \omega \end{aligned}$$

$A_2 = /l \,.\, \mathsf{sub}_{(z)}\Big(\mathsf{let}(\mathsf{letd}(\mathsf{var}_z) \mid \mathsf{letb}_{(y)}(S)) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val(i5)}))$. Next up is application of the substitution rule, (4.15). We have $\eta = \{0, 1 \mapsto 0\}$ and

$\vec{\iota} = (\mathsf{Id}_\emptyset, \mathsf{Id}_\emptyset)$.

$$
\begin{array}{rcl}
a & = & \mathsf{loc}_l : l \\
\omega & = & l/ \parallel \mathsf{id}_l : l \to (l, l) \\
C & = & /l \,.\, \mathsf{sub}_{(x)}\Big(\mathsf{let}(\mathsf{letd}(-_2\langle x, l\rangle) \mid \mathsf{letb}_{(y)}(S)) \mid -_3\langle x, l\rangle\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) \\
 & & \quad : (\{x, l\}, \{x, l\}) \to \emptyset \\
a' & = & \mathsf{loc}_l \parallel \mathsf{loc}_l : (l, l) \\
\omega' & = & \mathsf{id}_l \parallel \mathsf{id}_l : (l, l) \to (l, l)
\end{array}
$$

$A_3 = /l \,.\, \mathsf{sub}_{(z)}\Big(\mathsf{let}(\mathsf{letd}(\mathsf{val}(\mathsf{loc}_l)) \mid \mathsf{letb}_{(y)}(S)) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5})))$. Now rewrite A_3 with (4.6) and then (4.15) to obtain

```
A_5 = /l . sub_(z)(
            sub_(y)(snd(pair(
                          pairl(
                            assign(
                              aloc(val(loc_l)) |
                              aval(exp(val(i4))))))
                          pairr(exp(deref(var_z))))))
                    def_y(val(loc_l))) |
            def_z(val(loc_l))) |
          store(loc'_l(val(i5)))
```

These two steps unfolded the innermost let to a sub, and then substituted in the value of x for y. We see that there are no more free occurrences of the name y so we can terminate that substitution with (4.16). $\eta = \mathsf{Id}_0$.

$$
\begin{array}{rcl}
a & = & T \parallel \mathsf{val}(\mathsf{loc}_l) : (\{z, l\}, l) \\
\omega & = & \mathsf{id}_{\{z,l\}} \mid \mathsf{id}_l : (\{z, l\}, l) \to \{z, l\} \\
C & = & /l \,.\, \mathsf{sub}_{(z)}\Big(-_2\langle z, l\rangle \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) : \{z, l\} \to \emptyset \\
a' & = & T : \{z, l\} \\
\omega' & = & \mathsf{id}_{\{z,l\}} : \{z, l\} \to \{z, l\}
\end{array}
$$

where T denotes the sub-term snd(...) of A_5. $A_6 = /l \,.\, \mathsf{sub}_{(z)}\Big(T \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5})))$. Now, simply remove the exp from aval in T with (4.9) to obtain

```
A_7 = /l . sub_(z)(snd(pair(
                          pairl(assign(
                                  aloc(val(loc_l)) |
                                  aval(val(i4)))) |
                          pairr(exp(deref(var_z))))) |
                  def_z(val(loc_l))) |
          store(loc'_l(val(i5)))
```

A_7 can be rewritten with (4.10) as follows. $\eta = \{0 \mapsto 0, 2 \mapsto 2\}$.

$$\begin{aligned}
a &= \mathsf{i4} \parallel \mathsf{val}(\mathsf{i5}) \parallel 1 : (\emptyset, \emptyset, \emptyset) \\
\omega &= \mathsf{id}_\emptyset \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset) \\
C &= /l \,.\, \mathsf{sub}_{(z)}\Big(\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(-_3\langle l\rangle) \mid \mathsf{pairr}(U))\Big) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid -_4\langle l\rangle \\
&\quad : (l, l) \to \emptyset \\
a' &= \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset, \emptyset) \\
\omega' &= 1 \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset) \to (\emptyset, \emptyset)
\end{aligned}$$

$$\begin{aligned}
A_8 = /l \,.\, \mathsf{sub}_{(z)}\Big(&\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(U))\Big) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \\
&\mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))
\end{aligned}$$

Notice how the store has changed. It is here that we realise why the link l has to be open in the definition of (4.10). Had it been closed we would not be able to supply a proper context C because there would be no way to identify the location inside def_z with the locations inside R (that are plugged into the holes $-_3, -_4$), since these would be in an already closed link, and thus not accessible (by composition) from the outside. This is an important property of closed links. We may now rewrite A_8 to

$$\begin{aligned}
A_{11} = /l \,.\, &\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(\mathsf{deref}(\mathsf{val}(\mathsf{loc}_l))))\Big) \\
&\mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))
\end{aligned}$$

by the non-controversial sequence (4.2) - (4.15) - (4.16). Finally, we may rewrite using (4.8).

$$\begin{aligned}
a &= \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset, \emptyset) \\
\omega &= 1 \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset) \to (\emptyset, \emptyset) \\
C &= /l \,.\, \mathsf{snd}(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(-_3\langle l\rangle))) \mid -_4\langle l\rangle : (l, l) \to \emptyset \\
a' &= \mathsf{val}(\mathsf{i4}) \parallel \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset, \emptyset, \emptyset) \\
\omega' &= \mathsf{id}_\emptyset \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset)
\end{aligned}$$

Again, the rule would not have worked with a closed link had there been a dereference operation inside pairl of A_{11}, for example. In this case, however, it works out either way. We finish evaluation by trivially rewriting with (4.3) and then (4.5). Thus, we have $A_{14} = /l \,.\, \mathsf{val}(\mathsf{i4}) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))$ as promised.

4.4 Dynamic correspondence

We have argued by example that there is a dynamic correspondence between Ξ and the encoding hereof in local bigraphs. Informally, we claim that every time we reduce a term in Ξ by $\longrightarrow$ we can mimic that in the bigraphical reactive system with

one or more uses of $\longrightarrow$. To formally prove the dynamic correspondence we would need a substitution lemma, a lemma allowing us to separate a context from a redex, and a lemma corresponding to definition 4.3.1, but for bigraphs. Proving that the correspondence holds both ways would likely require us to keep track of which bigraphs are actually images of Ξ-terms. We leave this for future work.

We conclude that references can be encoded in local bigraphs using closed links, that they seem to behave as desired, and conjecture that this can be proved formally.

4.5 Discussion

One may wonder, intuitively, why we chose to encode references via closed links and not binders. We give such an intuition here. Closed links were included in the bigraph theory to capture the notion of *name restriction* from the π-calculus. Binding ports likewise capture the π-calculus notion of *prefix*. In essence, the difference between closed links and binding ports in (binding and local) bigraphs is *locality* and the *scope rule*. All binders are located (while edges are not) and have to obey the scope rule, i.e. all peers of a certain binder must be located beneath that binder in the place graph, roughly said.[3] This means that using binders is in general more restrictive, which may be preferred in some situations like when modelling *access control*. In our case we have a *global* store located at top level. Thus, in this case, it should be possible to encode references via a top-level vertex acting like a store by having a binding port for each created location. Summing up, we can say that closed links were chosen because they do not have locality, which enables us to avoid worrying about the locality of the store holding the references, which is the natural way to think about a store, we believe. A reason for using binders could be to investigate whether there is a simpler proof of the dynamic correspondence between Ξ and the encoding, than with free closed links. The idea is to define a family of controls store^m, indexed by an ordinal m defining the set of (binding) ports on store. The program resides within this control. Initially, we have $\mathsf{store}^0(\ldots)$ with no ports meaning that no locations have been created. Then, every time a location is created we replace the current store^i with a new one, namely store^{i+1}, and bind this new location to the new port (keeping the old bindings). When binding a new location to the store we also create a value node holding the value of the newly created location, and link this to the new binding port of the store. This idea resembles one of Robert Harper [Har00] (chapter 16), though in another setting.

[3] We aim to provide some intuition and thus refer the reader to [JM04] for precise definitions.

Chapter 5

A Bigraphical Location Model

5.1 Introduction

In this chapter we make more precise what is meant by the term "reflective" building. Having a clear idea of what properties such a building should possess we move on to exhibit a Plato-graphical model corresponding to such a building. Some parts of the model are formulated directly in bigraphs, and some parts are expressed in a slightly enhanced and sugared form of Ξ, named Ξ_{sugar}. In a sense, this chapter brings together the work of the chapters 2, 3, and 4. This chapter presents work in progress.

5.2 A reflective building

We define a reflective building in more detail. In [Hop00] on "sentient computing" the following three "location categories" are stated as important.

- Containment
- Proximity
- Coordinate systems

Containment refers to an object being within a container (location). Proximity is the notion of being close to something. Coordinate systems provide location in space (subject to some error value). We have seen containment before as a spatial relationship on locations. We interpret proximity to cover two situations: (1) Near measured in physical distance, and (2) near as within the same symbolic range, i.e. within the same container (at some level in the topology/hierarchy). Our starting point is a symbolic model so we abstract away from coordinates in this treatment.

We do, however, add some additional properties to our list of reflective building properties:

- Device positioning
- A fixed set of uniquely identifiable mobile devices

When considering a reflective building we think of located-objects as being mobile devices (and henceforth refer to them as such) such a mobile phones and PDAs.

In our modelling effort we abstract the reflective building by abstracting over location types (floors, wings, rooms etc.) and device types (PDA, mobile phone, laptop etc.).

We capture systems where, e.g., a user arrives at a museum and receives a mobile device to be used together with the positioning system and the location model at this museum.

5.3 The model

Before the presentation of the Plato-graphical model it is instructive to explain the key design choices made.

5.3.1 Design choices

Several choices have been made regarding the languages used, the representation of locations and devices, and the location hierarchy. We treat these considerations in turn.

5.3.1.1 Languages used

We have used two languages in the model. The parts **C** and **S** of the model are written using the bigraphical term language, and parts **L** and **A** in a sugared version Ξ_{sugar} of the calculus Ξ presented in chapter 4. We prefer to work within bigraphs except when it becomes too inconvenient. We show that bigraphs are well-suited for modelling **C** and **S**, and not just **C** as suggested in chapter 3. The location model **L** is, however, sufficiently complicated to write natively in bigraphs for us to prefer writing it in Ξ_{sugar}. **A** communicates with **L** so it makes sense to write **A** in Ξ_{sugar}, apart from the fact that it is easier to program **A** in Ξ_{sugar} than natively in bigraphs, as will become apparent. Thus, we have a real world **C** written in bigraphs, a proxy $\mathbf{P} = \mathbf{S} \parallel \mathbf{L}$ spanning both bigraphs and Ξ_{sugar}, and a location-aware application **A** written in Ξ_{sugar}. A key issue in this setup is how to realise the communication between the "bigraph world" and the "Ξ_{sugar} world" in **P**. Later in this chapter we explain how Ξ_{sugar}-programs correspond to Ξ-programs.

5.3.1.2 Representing locations and devices in C (and S)

We use one control $\mathsf{loc} : 0 \to 0$ to represent locations and one $\mathsf{dev} : 0 \to 0$ to represent devices abstracting away different location types. Another option is to use different controls for the "different" types of locations in a building; wing, floor,

room and so forth. The reasons for choosing one loc control are (1) to limit the number of reaction rules to be written – we do not need to have a rule for each combination of the different location (and device) types where one location is source and the other target, and (2) identity and type of a location can be represented conveniently in another way (to which we return below). Devices are simply represented by a device control.

5.3.1.3 Known devices

As mentioned in section 5.2 we consider a reflective building to have a fixed set of known devices, which pertains to all parts of the model. Some may be in use and some may not. A device is in use when it is in a location which is not the special "unused devices" location. Initially, each device is either in this location or in one of the locations of the location hierarchy (to be addressed next). This property should be invariant under reaction. No new devices can appear. As mentioned in section 5.2 this is a realistic choice, and technically it significantly simplifies our task. This is due to the fact that when a device is discovered in **C** we have to mirror this by a discovery in **L** via **S**, and thus generate a corresponding fresh identifier for this device in **L**. It is difficult to ensure this automatically.

5.3.1.4 A static tree-structured location hierarchy

It is a simplifying choice to work with a static location hierarchy as opposed to a dynamic one. It is reasonable to have a static location hierarchy in this case because a (reflective) building in the real world seldomly changes. Thus, the location hierarchy in the model needs to be altered only on rare occasions. According to [Leo98] location hierarchies are typically static. Furthermore, we organise locations in a tree utilising the structure of place graphs. This is a limiting choice, but not an uncommon one. The same hierarchy is present in **C** and **L**, but represented differently.

5.3.1.5 Modelling the real world

For our purposes the following abstraction is suitable: Devices can enter, move around in, and leave a (reflective) building which is observationally equivalent to being turned off. These reconfigurations should be modelled in **C**.

5.3.1.6 Identification of locations and devices in C and L

A location in the bigraphical part of the model can be identified by a link or by an embedded identity control. We assume that locations are uniquely identified which requires an equality operation on their identifiers. Such identifiers could, e.g., be natural numbers or strings. We have chosen natural numbers for simplicity and to keep our focus. Strings can also be encoded. The property that no two locations or devices have the same identifier is invariant under reaction, i.e. maintained by every reaction rule. Now consider the two approaches; identification via a link or an embedded

control. Links can be open (outer name) or closed (edge). Using closed links in **C** is not a viable solution because they do not in fact reveal any identity information to the context and thus can not be distinguished, which is required to support certain basic queries. Open links could be used because the location hierarchy is static; location identifiers are assumed to be unique initially, and new locations are not introduced under reaction. Thus, there is no risk of an identifier (outer name) being reused, i.e. two different locations being identified. As no new devices are introduced we are safe. The same reasoning applies to device identifiers. Another option is to place a unique identity control as a child of each location and device control, which is what we choose. Our choice is supported by the following two considerations: (1) The model becomes simpler. Had we chosen link identifiers in **C** we would have to relate these links to location identifiers in **L** which could be natural numbers (or strings). This can be done by "exporting" the location identifiers of **L** to top level of the Plato-graphical system, which is not entirely straightforward as will become apparent later. (2) Given the intuition that controls represent entities and links represent (wireless) connections between entities it seems appropriate to model identity of an entity as part of that entity (e.g. like a MAC address of a device or the name of a room), i.e. via an identity control. Initially, **C** and **L** have the same location hierarchy (in their respective languages). Next we discuss how closely coupled **C** and **L** should be.

5.3.1.7 Relating C and L via S

An argument for a high degree of coupling between **C** and **L** is a low modelling effort, whereas a low degree of coupling implies stronger modularity, which is an argument against a tight coupling. Let us consider the setup to decide on a design choice here. The setup is that the initial configuration of the system is given, and in particular that the location hierarchies in **C** and **L** are coherent. For simplicity, take locations and devices to be identified by Ξ-integers in **L**. Encoding integers (or rather natural numbers) in bigraphs yields an easy correspondence, defined in **S**, between identifiers in **C** and **L**. This is a rather tight coupling, but not an unreasonable one since the user of the system is obliged to define the location hierarchy and the devices in both **C** and **L** initially. Furthermore, decoupling **C** and **L** merely comes down to a mapping in **S**.

5.3.1.8 Location systems as Plato-graphical systems

Consider figure 5.1. Location-aware applications (agents) are captured by the part **A**. The location model including queries (and actuators) is part **L**. We have included both queries and actuators in **L** because they are really interfaces to the location model. The positioning system including sensors is part **S**. **S** informs **L** of location updates. The physical world is part **C**.

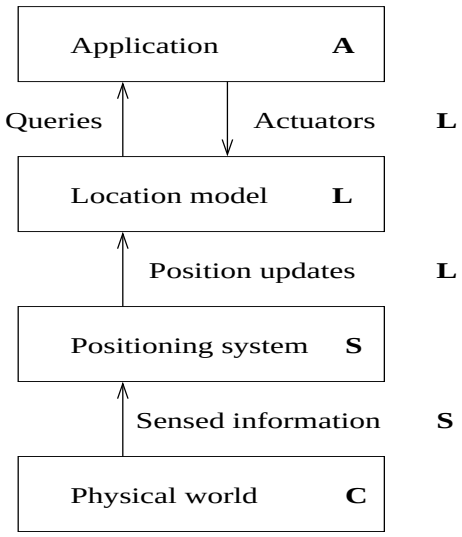


Figure 5.1: Overall location system model seen as Plato-graphical.

5.3.2 Introducing the model

We begin by briefly recalling Plato-graphical models. Consider a Plato-graphical model $\mathcal{X} = (\mathbf{C_X}, \mathbf{P_X}, \mathbf{A_X})$ as defined in chapter 3. This model consists of a world $c_X \in \mathbf{C_X}$, a proxy $\mathbf{P_X}$ with constituents $s_X \in \mathbf{S_X}$ (sensor) and $l_X \in \mathbf{L_X}$ (the location-aware system's representation of the world l_X), and a location-based application $a_X \in \mathbf{A_X}$. The overall intuition about the system is as follows: $c_X \in \mathbf{C_X}$ reconfigures as it pleases. Essentially, we have three reconfigurations of c_X; discovery of a device in some location, movement of a device from a location to a parent or sub-location, and loss of a device (the system loses track of the device). Much like in the real world, changes in c_X occur in an unpredictable and non-deterministic manner. The system s_X *observing* the state c_X of the world tries as best it can to *inform* l_X of change to c_X by invoking certain "interface functions" in l_X to update its internal representation of the world (building). s_X has access to c_X and l_X by shared controls (with c_X) and an outer name (with l_X). Due to the asynchronicity between c_X and $\mathbf{L_X}$ we are likely to experience some discrepancy in the "states" of the two parts. This is perfectly realistic wrt. real-life indoor positioning systems such as Ekahau. To make our model even more realistic we envision introducing time so that events from s_X to l_X can be timestamped and thus ordered. This should enable l_X to update its internal representation to more accurately match c_X as observed by s_X. a_X can query l_X via a specific set of "interface functions". Recall that $\mathbf{C_X}$ and $\mathbf{S_X}$ are native BRSs whereas $\mathbf{L_X}$ and $\mathbf{A_X}$ are the BRSs resulting from a translation of Ξ_{sugar}-programs into local bigraphs along with the bigraphical reaction semantics defined in chapter 4. Thus

utilising the “multi-lingual feature” of Plato-graphical models. Having briefly given the intuition behind the workings of the model we proceed to define it as a Plato-graphical model.

5.3.2.1 Our reflective building

We choose an example which holds the essential conceptual challenges arising from the previous discussion, but is also kept simple for the sake of clarity. A more complex building, e.g. the IT University of Copenhagen, can likely be modelled without problems.

Introducing the simple reflective building We model a piece of ITU. Before showing the building as a tree and a bigraph, we briefly state the intention. We model the building, the atrium (in reality spanning all five floors) in the centre of the building, two of the four wings, two of the six floors, a hallway, and two rooms. The building (i1) contains the atrium (i2) and the wings B (i3) and C (i4). Wing C spans two floors, namely the third floor (i5) and the fourth floor (i6). The fourth floor contains a hallway (i7) which in turn contains two rooms, namely room 4C16 (i8) and 4C10 (i9). We assume six known devices numbered i10 through i15, all in use. One device in the building, two devices are in the atrium, one is on the third floor, one is in room 4C16, and one in room 4C10. We could have made other choices, which we address shortly, but for now, we ask the reader to consider the informal graphical representation in figure 5.2. Do notice that the id-controls around location identifiers have been left out for simplicity. Some choices were made to arrive at exactly the location hierarchy of figure 5.2, each one is addressed in turn.

- The ordering of wings and floors.
- Where devices can reside.

We could, of course, have included more wings, floors, rooms etc. easily. First of all, notice how wings are considered to be above floors in the hierarchy. This indicates that one has to be within a wing to move from one floor to another. At the ITU we have stairways that allow this movement. However, it is also possible to move from a wing to the building, into an elevator, and to another floor. This is not possible in this model, but we consider it to a cosmetic problem that is irrelevant wrt. our purposes at present. Switching the ordering of wings and floors would yield a similar problem. Thus, we can either stick with an ordering and accept the limitation, or model the location hierarchy using two trees (views), one with wings above floors and the other with floors above wings. That would, however, complicate things because devices would then have to be situated in both trees and make a movement in both trees at once. This movement would be between siblings in one tree, but not in the other tree. As we argue below, the movement between siblings is the most reasonable and realistic having chosen a tree structure to organise locations. Alternatively, links could be used to indicate paths between locations, but that would defeat the purpose

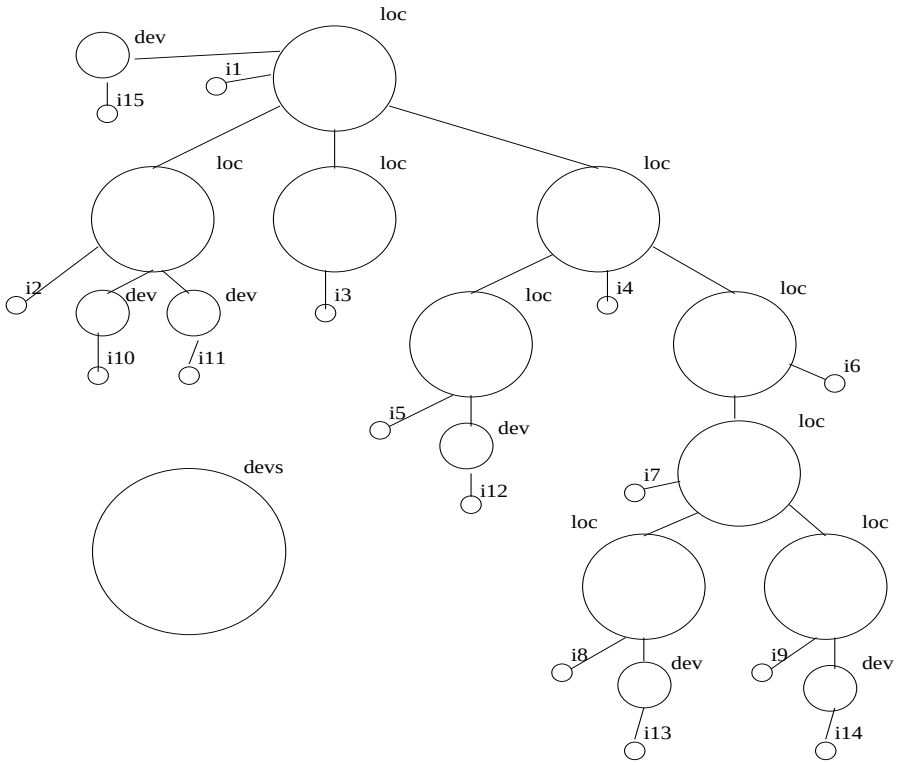


Figure 5.2: The building β as a tree where the id-controls around location identifiers have been left out, for simplicity.

of having imposed a structure on locations in the first place. A brief detour: The link graph is usable for supporting the connected-to and distance relationships of chapter 2 that allow for nearest neighbour and navigation queries.

We have decided that devices can reside in any location, and that locations can contain both devices and other locations. This is flexible and does not cause any modelling problems.

For comparison, the building β is depicted as a bigraph in figure 5.3. Again, notice that the id-controls around location identifiers have been left out for simplicity. We briefly comment on figure 5.3. Identifiers (natural numbers) are represented by nodes with atomic controls so we depict them as small black boxes. It should be clear that the bigraph corresponds exactly to the tree of figure 5.2. We proceed by presenting the model as a Plato-graphical system.

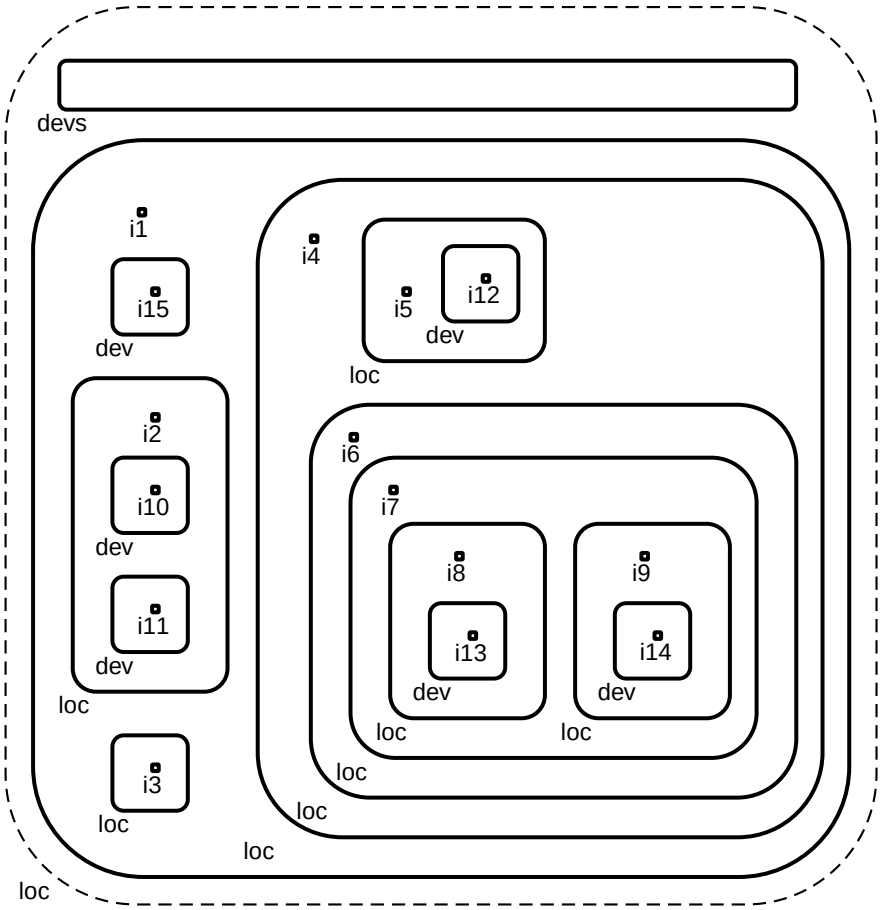


Figure 5.3: The building β as a bigraph in graphical representation where the id-controls around location identifiers have been left out, for simplicity.

5.3.3 $\mathcal{X}$ as a Plato-graphical model

We divide the presentation into four parts: The native bigraphical parts (1) $\mathbf{C}_\mathbf{X}$ and (2) $\mathbf{S}_\mathbf{X}$, and the Ξ_{sugar}-parts (3) $\mathbf{L}_\mathbf{X}$ and (4) $\mathbf{A}_\mathbf{X}$.

Recall that we work in local bigraphs so the arity of a control is *binding* $\rightarrow$ *free*. Furthermore, we consider *activity* and *atomicity* to be integral parts of bigraphs, but as shown in [Jen06] activity and atomicity could be considered *place-sortings* on *basic* bigraphs. We refer to [Jen06] for the details.

5.3.3.1 The world part $\mathbf{C_X}$

We begin by showing the building β as a bigraph in our term language (as before, omitting details such as identity wirings etc.), and then proceed to define the signature and dynamics of the BRS $\mathbf{C_X}$.

```
loc(i1 | dev(i15) |
      loc(i2 | dev(i10) | dev(i11)) |
      loc(i3) |
      loc(i4 | loc(i5 | dev(i12)) |
               loc(i6 | loc(i7 | loc(i8 | dev(i13)) |
                                 loc(i9 | dev(i14))))))
| devs()
```

As mentioned, β is a tree. Having made that choice it is reasonable to model device movement by allowing devices to move from a location l_2 into a sub-location l_3 of l_2, or into the parent location l_1 of l_2. To justify this claim, think of β; device i13 is situated in location 4C16 (i8), and the only doorway of 4C16 is into the hallway (i7), and the hallway does allow movement into another office, namely 4C10 (i9). All devices are in use in this start configuration. Remember that a device is either in a location or in devs.

The signature and dynamics of $\mathbf{C_X}$ are defined as follows in figure 5.4; $\mathbf{C_X} = (\mathcal{K}_{\mathbf{C_X}}, \mathcal{R}_{\mathbf{C_X}})$. First, notice how we have allowed ourselves to represent natural num-

Context $\mathbf{C_X}$.

Control	Activity	Arity	Comment
id	passive	$0 \to 0$	Hosts identifier control
loc	passive	$0 \to 0$	Possibly nested location
dev	passive	$0 \to 0$	Mobile device
devs	passive	$0 \to 0$	Hosts mobile devices not in use
i0,i1,i2...	atomic	$0 \to 0$	Infinite family of identifiers

$$\mathsf{loc}(-_0) \parallel \mathsf{devs}(-_1 \mid \mathsf{dev}(-_2)) \longrightarrow \mathsf{loc}(-_0 \mid \mathsf{dev}(-_2)) \parallel \mathsf{devs}(-_1) \tag{5.1}$$

$$\mathsf{loc}(-_0 \mid \mathsf{dev}(-_2)) \parallel \mathsf{devs}(-_1) \longrightarrow \mathsf{loc}(-_0) \parallel \mathsf{devs}(-_1 \mid \mathsf{dev}(-_2)) \tag{5.2}$$

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(-_1 \mid \mathsf{dev}(-_2))) \longrightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_1) \mid \mathsf{dev}(-_2)) \tag{5.3}$$

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(-_1) \mid \mathsf{dev}(-_2)) \longrightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_1 \mid \mathsf{dev}(-_2))) \tag{5.4}$$

Sorts:
(5.1) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{C_X}}$
(5.2) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{C_X}}$
(5.3) : $\mathcal{K}_{\mathbf{C_X}}$
(5.4) : $\mathcal{K}_{\mathbf{C_X}}$

Figure 5.4: Part $\mathbf{C_X}$ of the Plato-graphical model $\mathcal{X}$, with sorts.

bers by i−controls instead of using the more low-level representation with zero and

successor. Equality and other basic operations can be implemented by an infinite set of reaction rules, capturing all combinations. Rule (5.1) discovers a device by moving it from devs to some loc. The rule is parametrised over the identity of the location, its content before the discovery, the set of known devices, and the identity of the device being discovered. Rule (5.2) performs the opposite operation, namely to lose track of a device. Notice how a device can be discovered and lost in any location which allows us to model a device being switched off (manually or because the battery runs out, e.g.) and turned on again in another location by rule (5.1). The rules (5.1) and (5.2) are dual in a sense. The rules (5.3) and (5.4) are likewise dual and move a device up or down one step in the location tree. We argued earlier that this is a fair representation of movement having chosen a tree structure as location hierarchy. One may wonder why we do not simply have one rule to move a device from one location to any other since we do not impose any restrictions on movement. There are two reasons for that: (1) It contradicts the choice of a tree structure of locations, and (2) we would need two rules. We believe the first point has been covered already and proceed to justify the second. Consider the following rule.

$$\mathsf{loc}(-_0 \mid \mathsf{dev}(-_1)) \parallel \mathsf{loc}(-_2) \longrightarrow \mathsf{loc}(-_0) \parallel \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))$$

This wide rule is not what we want. It can, as expected, perform the following reaction:

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(\mathsf{dev}(-_1))) \mid \mathsf{loc}(-_2) \longrightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}()) \mid \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))$$

More generally, it can move a device from one location into another location provided that there is a context which assigns them a *common parent*, which is not one of the locations in question. However, it can *not* perform the following reaction:

$$B = \mathsf{loc}(-_0 \mid \mathsf{dev}(-_1) \mid \mathsf{loc}(-_2)) \longrightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))) = B' \quad \not\downarrow$$

That is, it can not move a device from a location l_1 to a sub-location of l_1. To realise this let us try to construct a match; it suffices to argue that for all C we have $B \neq C \circ (R \oplus \omega) \circ a$:

$$\begin{aligned} R &= \mathsf{loc}(-_0 \mid \mathsf{dev}(-_1) \parallel \mathsf{loc}(-_2)) : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset) \\ a &= \mathsf{i1} \parallel \mathsf{i2} \parallel \mathsf{i3} : (\emptyset, \emptyset, \emptyset) \\ \omega &= (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) \parallel \mathsf{id}_\emptyset : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset) \\ C &= \mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset : (\emptyset, \emptyset) \to \emptyset \end{aligned}$$

Clearly, this is not a match, and can not be made so. The intuition is that the context C must have two holes because R has two regions. Intuitively, the trouble is that the context can not take something from one of these holes and put it into the other – it sees the regions of R (and R') from above, so to speak. One could try to circumvent this mechanism by including one location in the other through a parameter a, but that also fails, because the context can not get rid of either one of the parameters it absorbs, and will thus have one location too many for the match. Thus, for the

scheme with the wide rule to work as intended, we must include another rule that can move a device into a sub-location. We have such a rule above, namely (5.4) which can be applied consecutively. We conclude that the two movement rules chosen, (5.3) and (5.4), are the better choice.

With this analysis in mind, we remark that the suggested rule (3.10) of chapter 3 perhaps should be replaced by two rules.

This concludes our treatment of $\mathbf{C_X}$. We proceed with $\mathbf{S_X}$ and then $\mathbf{L_X}$ before gluing together $\mathbf{C_X}$ and $\mathbf{L_X}$ via $\mathbf{S_X}$.

5.3.3.2 The sensor part $\mathbf{S_X}$

$\mathbf{S_X}$ is the representation of a simple sensor system that can (1) observe (sense) that a device is in a certain location (in c_X), and (2) observe that a device is not located (sensed), i.e. residing in "location" devs. There could be (at least) two reasons for losing track of a device: (1) It was turned off, and (2) the positioning system simply did not pick up on the signal from the device (for a certain amount of time). Recall that the set of devices is fixed and known. s_X informs l_X about these observations by invoking certain "interface functions" provided by l_X. We will examine this communication in subsubsection 5.3.3.3.

Because we have chosen a rather tight coupling between c_X and l_X, namely to represent location and device identifiers by the same natural numbers in both worlds, s_X does not need to maintain a mapping from one world to the other. We do require that Ξ-integers are represented in the same way as integers in $\mathbf{C_X}$ – namely by i-controls. s_X is written in bigraphs, but also has access to bigraphical *representations* of Ξ- terms.

The signature and dynamics of $\mathbf{S_X}$ are defined as follows in figure 5.5; $\mathbf{S_X} = (\mathcal{K}_{\mathbf{S_X}}, \mathcal{R}_{\mathbf{S_X}})$. Rule (5.5) models the case where s_X observes a device in a location, and informs l_X of this. This is done by "calling" the function in l_X, exported by g as the first component in a tuple, with the bigraphical representations of the device and the location. invoke models a pool of pending function calls and must have name $funs$ and be empty in the initial configuration of s_X. In the rule we have that var_g refers to exactly the same name as invoke_g to force the var-control to refer to precisely the name exported from L_x. To fire the context of the rule must identify g and $funs$. We need to encapsulate the location identifier in a id-control to distinguish it from sublocations and thus make the rule work for any location (identifier) $-_0$. Rule (5.6) applies when s_X observes a device inside the devs control. We will show the relevant functions in subsubsection 5.3.3.3.

Notice how s_X merely observes c_X (sensing) and informs l_X (acquisition). This may very well lead to discrepancy between c_X and l_X. To make the system more precise, i.e. to ensure a tighter correspondence between c_X and l_X one could allow s_X to also observe l_X, and then only inform l_X of the location of a device c_X when c_X and l_X disagree. This is not the way positioning systems work in reality, but may be useful for simulation purposes. For now, we leave s_X as is. This concludes our treatment of the positioning system $\mathbf{S_X}$.

Sensor $\mathbf{S_X}$.

Control	Activity	Arity	Comment
id	passive	$0 \to 0$	Hosts identifier control
loc	passive	$0 \to 0$	Possibly nested location
dev	passive	$0 \to 0$	Mobile device
devs	passive	$0 \to 0$	Hosts mobile devices not in use
app	active	$0 \to 0$	Application
appl	active	$0 \to 0$	Left part of application
appr	active	$0 \to 0$	Right part of application
var	atomic	$0 \to 1$	Variable
exp	passive	$0 \to 0$	Delay evaluation
i0,i1,i2…	atomic	$0 \to 0$	Infinite family of identifiers
invoke	active	$0 \to 1$	Hosts "function calls"

$$\mathsf{loc}(\mathsf{id}(-_0) \mid \mathsf{dev}(-_1) \mid -_2) \parallel \mathsf{invoke}_g(-_3)$$

$$\longrightarrow$$

$$\mathsf{loc}(\mathsf{id}(-_0) \mid \mathsf{dev}(-_1) \mid -_2) \parallel \mathsf{invoke}_g(-_3 \mid \mathsf{app}(\mathsf{appl}(\mathsf{app}(\mathsf{appl}(\mathsf{fst}(\mathsf{var}_g)) \mid \mathsf{appr}(\mathsf{exp}(-_0)))) \mid \mathsf{appr}(\mathsf{exp}(-_1)))) \tag{5.5}$$

$$\mathsf{devs}(-_0 \mid \mathsf{dev}(-_1)) \parallel \mathsf{invoke}_g(-_2)$$

$$\longrightarrow$$

$$\mathsf{devs}(-_0 \mid \mathsf{dev}(-_1)) \parallel \mathsf{invoke}_g(-_2 \mid \mathsf{app}(\mathsf{appl}(\mathsf{fst}(\mathsf{snd}(\mathsf{var}_g))) \mid \mathsf{appr}(\mathsf{exp}(-_1)))) \tag{5.6}$$

Sorts:
(5.5) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{S_X}}$
(5.6) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{S_X}}$

Figure 5.5: Part $\mathbf{S_X}$ of the Plato-graphical model $\mathcal{X}$, with sorts.

5.3.3.3 The location model part $\mathbf{L_X}$

This part is implemented in Ξ_{sugar} because writing it natively in bigraphs proved too cumbersome. We saw the "findall" query in appendix B of chapter 3, and it is significantly more involving to encode more advanced queries. Here we explain the ideas involved in this part of the model, and in subsection 5.3.4 we explain how to go from a Ξ_{sugar}-program to a Ξ-program automatically. The presentation of L_x is divided into the following parts:

- Data-type declarations
- The building configuration
- The interface to $\mathbf{S_X}$; reconfigurations

- The interface to $\mathbf{A_X}$; location-based queries
- Communication between $\mathbf{L_X}$ and $\mathbf{S_X}$

When these pieces are in place we glue them together to form a presentation of (the structure of) L_x as a whole. To keep the discussion focused we take the liberty of being very brief when treating auxiliary functions in a_X – we are interested in the reconfigurations of l_X and the queries supplied to a_X.

Data-type declarations are a convenient way to abstract away from the underlying type when programming in (a fraction of) SML. In L_x we choose to work with just one basic data-type for location and device identifiers; natural numbers. Natural numbers and equality on them can be easily encoded in bigraphs, as seen in chapter 4. We could have used strings instead, but natural numbers are simpler and suffice for our purposes.

The data-types used in L_x are as follows.

```
type lid = int
type dev = int
datatype hierarchy = (* id, devices, sublocations *)
         Loc of lid * dev list * hierarchy list
```

A hierarchy represents the aforementioned location tree. A location has an identifier, a list of devices, and a list of sub-locations. Both types and data-types belong to the set $\mathcal{C}$. For now, we assume lists as a predefined data-type. We also assume a few basic operations on lists, namely cons ('::'), append ('@'), reverse ('rev'), and 'map'. We address how to encode lists and the associated operations in Ξ in subsection 5.3.4.

The building configuration The building looks like this in Ξ_{sugar} and corresponds exactly to the one in $\mathbf{C_X}$. The initial configuration l_X is shown below along with some enclosing Ξ_{sugar} code.

```
val funs =
  let val state =
    ref (Loc(1,[15],
             [Loc(2,[10,11],[]),
              Loc(3,[],[]),
              Loc(4,[],
                  [Loc(5,[12],[]),
                   Loc(6,[],
                       [Loc(7,[],
                            [Loc(8,[13],[]),
                             Loc(9,[14],[])])])])]))
      val devs = ref []
```

```
    ...
in ... end
```

As can be seen, we implement devs as (a reference to) a list. We use references to update the internal representation of l_X. The Ξ-locations are translated into bigraphs by the translation given for constructors in chapter 4. The dynamics of $\mathbf{L_X}$, i.e. reconfigurations of this configuration are implemented by Ξ-functions, to which we will return later. Three dots signify place-holders, ignore them for now. We explain what happens to the 'val' declaration in subsection 5.3.4.

The interface to $\mathbf{S_X}$; reconfigurations $\mathbf{L_X}$ supplies $\mathbf{S_X}$ with some interface functions. The purpose of these functions is to allow $\mathbf{S_X}$ to inform $\mathbf{L_X}$ of events (movement of devices), which $\mathbf{S_X}$ has observed in $\mathbf{C_X}$. As mentioned, $\mathbf{S_X}$ can do two things:

- Observe that a device is in a certain location, and inform $\mathbf{L_X}$.
- Observe that a device is not sensed in any location, i.e. currently not in use and residing in "location" devs, and inform $\mathbf{L_X}$.

These two situations give rise to two reconfigurations, i.e. Ξ_{sugar}-functions.

```
fun sobserved d =
    fn l =>
       let val state' = delete d (!state)
           val devs' = del_list d (!devs)
           val state'' = insert d l (!state)
       in state:=state''; devs:=devs' end

fun slost d =
    let val state' = delete d (!state)
        val devs' = del_list d (!devs)
        val devs'' = d::devs'
    in state:=state'; devs:=devs'' end
```

The function `sobserved` is called by s_X (we will return to what a "function call" means) when s_X observes that device `d` is in location `l` in c_X. The function makes sure to delete `d` everywhere from the `state` hierarchy l_X, and then inserts it in the (possibly) new location `l`. Like in c_X this guarantees that `d` is either in exactly one location or in `devs` at any point in time. This is under the assumption that it is called with an existing location. The function `slost` is similar, but places `d` in `devs`. The auxiliary functions `delete`, `del_list`, and `insert` are functions internal to l_X, and `state` and `devs` are two private data structures in l_X constituting the current configuration. We show these functions below for completeness.

```
(* remove an element from a list *)
fun del_list e =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs

(* delete device 'dev' from hierarchy 'id' *)
fun delete dev =
    fn (Loc(id,ds,ls)) =>
       Loc(id, del_list dev ds, map (delete dev) ls)

(* insert device 'dev' into location 'lname'
   in hierarchy 'id' *)
fun insert dev =
    fn lname =>
       fn (Loc(id,ds,ls)) =>
          if lname=id then Loc(id,
                               dev::ds,
                               map (insert dev lname) ls)
          else Loc(id, ds, map (insert dev lname) ls)
```

They should be straightforward and the comments sufficient. We do, however, note that the functions are curried and thus take exactly one parameter. Also, notice how s_X does not know the internals of l_X, but merely the names of the two interface functions 'sobserved' and 'slost'.

The interface to $\mathbf{A_X}$; location-based queries The interface to $\mathbf{A_X}$ is currently defined by two functions visible to $\mathbf{A_X}$, but with internal workings local to $\mathbf{L_X}$. It is possible for the application ($\mathbf{A_X}$) to enquire to the whereabouts of a certain device, and to enquire about all the devices to be found in a certain location.

```
fun awher d = whr d (!state)
fun afind lname = flocs (pickloc lname (!state))
```

These functions have no side effects, in particular they do not alter l_X. The function 'awher' is based on the following auxiliary functions, which should be read bottom-up.

```
(* find the identifier of a device's location *)
fun whr dev =
    fn l =>
       case l of
           (Loc(_,[],[])) => NONE
         | (Loc(id,d::ds,ls)) =>
           if dev=d then SOME(id)
```

```
          else whr dev (Loc(id,ds,ls))
        | (Loc(_,[],ls)) =>
          let fun whr' =
                  fn list =>
                     case list of
                        [] => NONE
                      | (loc::locs) =>
                           case whr dev loc of
                              SOME(i) => SOME(i)
                            | NONE => whr' locs
          in whr' ls end
```

We believe that the comments are sufficient explanation. We address 'NONE', 'SOME(x)' (which can also be used in $\mathbf{A_X}$) and embedded functions in subsection 5.3.4. The function 'afind' uses the following auxiliary functions.

```
(* find all devices in a hierarchy - depth first *)
fun fall (Loc(_,ds,[])) = ds
  | fall (Loc(_,ds,l::ls)) =
    let fun fall' [] = []
          | fall' (loc::locs) = (fall loc) @ (fall' locs)
    in ds @ (fall l) @ (fall' ls) end

(* pick the subtree with id 'loc' from a hierarchy *)
fun pickloc lname =
    fn (Loc(id,ds,ls)) =>
       if lname=id then SOME(Loc(id,ds,ls))
       else let fun pickloc' [] = NONE
                  | pickloc' (loc::locs) =
                    case pickloc lname loc of
                        SOME(l) => SOME(l)
                      | NONE => pickloc' locs
            in pickloc' ls end

(* unpack option, return list of devices *)
fun flocs option =
    case option of NONE => []
                 | SOME(l) => fall l
```

Again, we refer to the comments for explanation.

Communication between $\mathbf{L_X}$ and $\mathbf{S_X}$ We promised to address the question of what it means to call a Ξ_{sugar}-function (in this case living in $\mathbf{L_X}$) from a native BRS (in this case $\mathbf{S_X}$). We need to be able to somehow export selected function names from $\mathbf{L_X}$ to

$\mathbf{S_X}$. Such a mechanism is not present in Ξ, so we introduce it by means of additional syntax. The effect is that L_x has the following form:

```
export <name> from <exp>
```

The idea is to export the names of some functions from L_x to other parts of the system. One can think of it as a special form of `val f = exp` known from SML. This is done by making a particular name global in the Plato-graphical system (by closure), and then to make sure that the translation of relevant functions in 'exp' in L_x refer to this name. The translation of this new syntactic construction is:

$$[\![\text{export } f \text{ from } exp]\!]_X = \mathsf{def}_f([\![exp]\!]_\emptyset)$$

We emphasise the fact that we export one name f, which can possibly name a tuple of function names (as is the case for l_X). Thus, we can use projections on the name ('val') f to refer to the individual function names of the tuple (nested pairs). This was done in $\mathbf{S_X}$ where used an explicit projection on a name g that must match the exported name f.

The Plato-graphical system with function export has the form

$$/f \,.\, \mathbf{C_X} \parallel \mathbf{S_X} \parallel \mathsf{def}_f([\![exp]\!]_\emptyset) \parallel [\![A_X]\!]_f$$

where A_X is the component $\mathbf{A_X}$ before translation into bigraphs. Notice that exp is translated from Ξ_{sugar} into bigraphs using an empty set. This ensures that $f \notin fv(exp)$. f will be referred to from $\mathbf{S_X}$. For example, f could be `funs` and export a tuple of function names `sobserved` and `slost`.

It is important to be aware of the fact that $\mathbf{S_X}$ accesses the function 'sobserved' etc. by linking controls (var*s* in this case) to the projections on the exported name f. As we saw in the treatment of $\mathbf{S_X}$ above, $\mathbf{S_X}$ can produce function calls in $\mathbf{L_X}$ by introducing applications of var controls to (bigraphical representations of) arguments. We presented $\mathbf{L_X}$ in a form where functions take exactly one argument to match the way $\mathbf{S_X}$ introduces function calls.

To sum up, L_x has the following form:

```
export funs from
   let val ...
       fun sobserved d = fn l => ...
       fun slost d = ...
       ...
   in (sobserved, slost, ...) end
```

This concludes our treatment of inter-component communication between $\mathbf{S_X}$ and $\mathbf{L_X}$.

$\mathbf{L_X}$ collected We present the structure of L_x ($[\![L_x]\!] = \mathbf{L_X}$) as a whole, but we leave out (denoted by three dots) the auxiliary functions and comments, for readability.

```
export funs from
  type lid = int
  type dev = int
  datatype hierarchy =
           Loc of lid * dev list * hierarchy list
  ...
  let val state =
        ref(Loc(1,[15],
                [Loc(2,[10,11],[]),
                 Loc(3,[],[]),
                 Loc(4,[],
                     [Loc(5,[12],[]),
                      Loc(6,[],
                          [Loc(7,[],
                               [Loc(8,[13],[]),
                                Loc(9,[14],[])])])])]))
      val devs = ref []
      fun sobserved d =
          fn l =>
            let val state' = delete d (!state)
                val devs' = del_list d (!devs)
                val state'' = insert d l (!state)
            in state:=state''; devs:=devs' end
      fun slost d =
            let val state' = delete d (!state)
                val devs' = del_list d (!devs)
                val devs'' = d::devs'
            in state:=state'; devs:=devs'' end
      fun awher d = whr d (!state)
      fun afind lname = flocs (pickloc lname (!state))
  in (sobserved,slost,awher,afind) end
```

5.3.3.4 The signature $\mathcal{K}_{\mathrm{L_X}}$ for $\mathbf{L_X}$

Apart from the controls given in figure 4.1 we obtain the following controls by the translation given in chapter 4. They are shown in figure 5.6. This concludes out treatment of $\mathbf{L_X}$.

5.3.3.5 Dynamic correspondence between $\mathbf{C_X}$ and $\mathbf{L_X}$

$\mathbf{C_X}$ and $\mathbf{L_X}$ uphold the same invariants:

Model $\mathbf{L_X}$.

Control	Activity	Arity	Comment
Loc	active	$0 \to 0$	Location hierarchy constructor
Nil	passive	$0 \to 0$	Empty list constructor
Cons	active	$0 \to 0$	List constructor
NONE	passive	$0 \to 0$	Empty option constructor
SOME	active	$0 \to 0$	Option constructor

Figure 5.6: Additional controls for the signature of $\mathbf{L_X}$.

- A device is either in devs or exactly one location at any time.
- Locations and devices are uniquely defined, and in one-to-one correspondence between $\mathbf{C_X}$ and L_x.

We remark that it is reasonable to extend L_x with a parent map without doing so in $\mathbf{C_X}$ because the internal representation and data structures in L_x are of no concern to the real world as long as the invariants are kept. We stress the fact that these two worlds must agree on the representation of location identifiers, i.e. natural numbers. $\mathbf{C_X}$ uses i−controls, and so does $\mathbf{L_X}$ because of the translation defined. It is, however, up to the specifier to make sure that the same numbers are given to corresponding locations in both worlds.

5.3.3.6 The application part $\mathbf{A_X}$

This part should be a simple Ξ_{sugar}-program that utilises the queries supplied by L_x. An example application could be "find the nearest printer" wrt. the current location of my mobile device. We do not acutally give such a program here, but merely state that such a program should be straightforward to construct given what we have done in L_X.

5.3.4 From Ξ_{sugar} to Ξ

Ξ_{sugar} is a sugared version of Ξ which additionally has a mechanism to export function names to top level of Plato-graphical systems, as explained earlier. Most constructs in Ξ_{sugar} can simply be unfolded to or encoded as Ξ-constructs, but we also have the name-exporting enhancement adding modelling power to the calculus.

We begin with the enhancement. This is the main difference between Ξ_{sugar} and Ξ. The effect is that Ξ_{sugar}-programs p' are not just expressions, but are now encapsulated:

$$p' ::= \text{export } f \text{ from } p$$

The rest of the constructions are syntactic sugar. They are:

- Comments

- 'type' and 'val' constructs
- Nested 'let' construct
- Tuples
- Anonymous and nominal function declarations
- 'if-then-else' conditionals
- Basic operations on natural numbers
- Patterns

We treat each item in turn.

Comments are enclosed by '(*' and '*)' are simply discarded when translating L_x into bigraphs.

'type' and 'val' are handled as follows. type declarations are treated as textual substitution on the source program such that `type dev = int` is discarded and `dev` is replaced by `int` everywhere in the source program, for example. This, of course, requires that 'dev' is not used a variable name, i.e. $\mathcal{V} \cap \mathcal{C} = \emptyset$, as mentioned earlier. This means that any constructor name from $\mathcal{C}$ is legal in a Ξ_{sugar}-program, and that it is the programmers responsibility to use the so constructed terms correctly, i.e. to make sure that case expressions on a custom constructor have the right number of branches corresponding to the constructor.

Let us consider booleans and lists to see how the general constructor and case terms of Ξ_{sugar} can be represented in Ξ. For booleans the following terms are legal: True unit, False unit, and case e of True $x_1 \Rightarrow e_1$ | False $x_2 \Rightarrow e_2$. For lists: Nil unit, Cons (x_1, x_2), and case e of Nil $x \Rightarrow e_1$ | Cons $x \Rightarrow$ let h = fst x in let t = snd x in e_2. We would want to introduce abbreviations for nullary constructors; True, False, and Nil. Furthermore, we could introduce syntactic sugar for the case construct on lists: case e of Nil $\Rightarrow e_1$ | Cons $(h, t) \Rightarrow e_2$. As an example we show the semantics of lists (in Ξ and Ξ_{sugar}), which are more interesting than for booleans, would then be (leaving out the store since there is no side-effect):

$$\begin{aligned}
&\text{case Nil of Nil} \Rightarrow e_1 \mid \text{Cons } x \Rightarrow \text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2 \\
&\quad \longrightarrow \quad e_1 \\
&\text{case Cons } (v_1, v_2) \text{ of Nil} \Rightarrow e_1 \mid \text{Cons } x \Rightarrow \text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2 \\
&\quad \longrightarrow \quad (\text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2)\{(v_1, v_2)/x\}
\end{aligned}$$

The operations (functions) in lists, namely '::', '@', 'rev', and 'map' can clearly be coded in Ξ_{sugar} (and Ξ). The two functions '::' and '@' are used infix, but can easily be made prefix.

We have just seen how to represent booleans and lists. Another used data-type in L_x is options, i.e. 'NONE' and 'SOME(x)'. Having presented lists we trust that the reader is convinced that options can be represented similarly.

A 'val' declaration inside a 'let' expression is discarded, so `let val x = exp1 in exp2` becomes `let x = exp1 in exp2`.

Nested 'let' constructs can simply be unfolded. Thus,

```
let val x = ...
    val y = ...
in ... end
```

becomes

```
let val x = ... in
  let val y = ... in ... end
end
```

where the 'val' and 'end' keywords are discarded under translation.

Tuples are simply nested pairs such that $(a, b, c) = (a, (b, c))$ and so forth.

Anonymous and nominal function declarations We use two different ways of declaring functions in L_x; anonymous and nominal. The anonymous functions are of form `fn x => e` and are translated into $\lambda x.e$. Nominal functions on form `fun f x = e ...`, where the dots signify the rest of the program, are translated into `let f = (fix f(x) = e) in ...` because they can be recursive.

'if-then-else' conditionals are simply unfolded using the 'case' construct so `if b then e1 else e2` becomes `case b of True => e1 | False => e2`.

Basic operations on natural numbers are assumed to exist as primitives in Ξ_{sugar} because we have already shown that we can encode them in bigraphs. The operations used are '=', '¡=' (less than or equal to), and '-' (subtraction) on natural numbers.

Patterns are used heavily in L_x. It is perhaps easiest to see how patterns are unfolded by seeing and example. Consider the function `del_list` shown above:

```
fun del_list e =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs
```

First, unfold the 'fun':

```
val del_listl = fix del_list(e) =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs
```

Then, unfold the pattern:

```
val del_listl = fix del_list(e) =
    lambda y. case y of [] => []
                   | Cons z => let x = fst z in
                               let xs = snd z in
                               if e=x then del_list e xs
                               else x :: del_list e xs
```

where 'lambda' signifies λ. This Ξ_{sugar}-expression can be translated into a Ξ-expression by the methods shown above.

There are also more advanced patterns in use, i.e. patterns where we do not match merely a constructor like 'List', but a composite constructor such as 'Loc(5,Nil,Nil)'. Consider this case:

```
case exp of Loc(5,Nil,Nil) => e
```

We unfold this into

```
case exp of Loc p =>
  let a = fst p in
  let b = fst (snd p) in
  let c = snd (snd p) in
  e
```

adhering to the restriction that the left-hand side of a 'case' branch must consist of a constructor and a variable.

This concludes our justification of the claim that Ξ_{sugar}-programs can be translated into local bigraphs.

5.3.5 The model $\mathcal{X}$ is Plato-graphical

Having explained the model we state and state that it is actually Plato-graphical. This proposition relies on the fact that a given implementation of $\mathbf{A_X}$ uses a subset of the control of our $\mathbf{L_X}$ as given above.

Proposition 5.3.1. *The model $\mathcal{X}$ is Plato-graphical.*

Proof. It is enough to observe that $\mathbf{C}_X \perp \mathbf{A}_X$, and that $\vec{x} = funs$.

This concludes our presentation of the bigraphical location model. We proceed by relating the presented model to the location models of chapter 2.

5.3.6 Relating our model to the location model classification

In chapter 2 we introduced and classified location models. We justify our claim that our model is representative for a class of the presented models presented in chapter 2. It is important for our model to be representative because that property renders our work relevant for a wide range of location models, and thus a wide range of location systems.

5.3.6.1 Classifying the model

Clearly, the model $\mathcal{X}$ is symbolic, but not geometric. Thus, there is no notion of metric distance. We have shown that the location model L_x supports location queries and certain reconfigurations, and also the accumulating "find all" query, among others. We have not shown how to support range and navigation queries, but these queries can be easily programmed by introducing an explicit parent map to the location hierarchy. A range query is then a little primitive in the sense that ranges are determined by location containment. One could consider adding weights to edges and thereby enable shortest-path navigation queries. We have implemented these queries, but do not show them here because they do not add anything conceptual to this treatment. Some non-trivial work still remains to support "nearest neighbour" queries, namely to instrument the model with geometric information. To this end we could perhaps instrument each location with a coordinate control holding an ordered triple of integers.

We propose to look at the location hierarchy and queries supported for deciding when a given model correctly implements a specification. We conclude that the model is representative in the sense that it captures the essentials of an exclusive symbolic location model.

5.4 Concluding remarks

First, we consider whether we are any closer to answering the five questions of chapter 3:

1. What languages $\mathcal{L}$ can we encode?
2. How close are Plato-graphical models to real systems?
3. What challenges have we found for bigraphical models?
4. What uses do we envision for Plato-graphical models?

5. How do we reason about Plato-graphical models?

Ad. 1. We have successfully encoded an extended version of MiniML, which should to enable us to encode a wide range of location-aware applications. It turns out that reaction rules are enough for $\mathbf{C}_\mathbf{X}$ (and $\mathbf{S}_\mathbf{X}$).

Ad. 2. We have enough structure to represent an exclusive, symbolic location model. We have found possible uses for DAGs, timed and probabilistic events, and continuous space. Such extensions can be used to lift the model to real-life systems.

Ad. 3. We found that one may use closed links in a clever way (see chapter 3) to handle that something is *not* present in the context under reaction. We do, however, envision that this so-called "negative" context information will be needed in the bigraph theory in the long run, and conjecture that a safe sorting exists to enforce this.

Ad. 4. We envision to implement Plato-graphical models according to specifications of location models (and also context models) when a tool allows us to perform automated reactions. This is the basis of the simulation challenge.

Ad. 5. This is still an open question. One question is: To what extend can we transfer Ξ reasoning to bigraphs? As an example we could mention contextual term equality. That is if two Ξ-terms f and g are contextually equal, then are their images under $[\![\cdot]\!]$ contextually equal? One could argue that this question should be studied in a simpler and better understood framework than Plato-graphical systems. Another questions is: What does it take for us to be able to substitute one component for another in a Plato-graphical system (recall proposition 3.4.1 and definition 3.4.4 of chapter 3)? This is indeed an important question for future work.

5.4.1 Conclusions on our modelling effort

We draw the following conclusions:

- We can encode an extended version of MiniML in (local) bigraphs.
- We can represent an exclusive symbolic location model and all the desired query types on it in bigraphs. (We ask the reader to trust that we can program a symbolic range and navigation query since we have not shown these explicitly.)
- We can represent the world and a simple positioning/sensor system in bigraphs.
- We have taken one more step in evaluating Plato-graphical systems and thereby bigraphs as modelling formalism for GUC.
- We have argued that Plato-graphical systems enable convenient *modelling* of location-aware systems, i.e. facilitate programming of a location-aware application in Ξ_{sugar} querying a location model.

- The modelling effort was not low, but we now have a fairly accurate base system to support many other agents (location-aware applications).

Chapter 6

Related Work

6.1 Introduction

In this report we have taken an *experimental* approach to testing whether bigraphs are useful for modelling and programming context-aware systems. In this chapter we assess to what degree other formalisms can answer the challenge of modelling and programming context-aware systems.

Bigraph theory has roots in process calculi and in particular reactive systems (RSs), but is formulated in category theory. Thus, we consider formal approaches to explicit context-awareness based on process calculi and reactive systems to be closely related work. We also review a logic for specifying context-aware systems, which has a tuple space-based middleware supporting it. Due to the completeness (from theory to practice) of this approach we merit the presentation of it ample space.

We do *not* consider *algebraic graph transformation* in this report because there has, to the best of our knowledge, not been attempts to formalise context-awareness via algebraic graph transformation. It is, however, related work regarding bigraphs, so relating bigraphs and Plato-graphical models to algebraic graph transformation will be relevant, at least from a theoretical point of view, at some point. There are also several toolkits and middlewares available to support the implementation of context-aware systems, and some systems have such support built into them. We refer to [JPR04] for an overview of these as we do not in this piece of research consider implementations. Studying such systems in more detail could become relevant when implementing a context toolkit on top of a bigraphical rewrite engine. In our work we do not try to develop new location models and thus do not consider work in that area to be related, i.e. relevant for this chapter. We have merely provided a digest of that literature in chapter 2 of this report.

Having narrowed the scope of what we consider related work in this report, we proceed to give a method for treating the selected works.

6.1.1 Method

We treat each piece of related work in turn. For each piece of related work we take the following approach:

1. Report on the purpose/aim of the work and its strenghts and weaknesses. Include in this report the aspects that are important for modelling and programming context-aware and location-aware systems. We aim to be objective.
2. Evaluate the work wrt. how the formalism can be used to model and program context-aware (location-aware) systems:
   - Negative context information.
   - Control structures.
   - Representing a location topology.
   - Queries on the topology.
   - Interaction between a location-aware application and the location topology.
   - The modelling effort.
3. Reasoning in practise.

The first item is natural. We try to tailor the treatment of each piece of related work such that we can draw on it when relating the work to our own. We have pinpointed six items that are important to consider when estimating a formalism for modelling and programming location-aware systems, and location models in particular. This focus on the location aspect of context reflects the focus of our own work, and we are aware of the fact that some of the related works treated here capture a more general notion of context. The third item is about reasoning which is one of the main reasons for taking a formal approach to the study of context-aware systems. We emphasise that we wish to reason about concrete systems, i.e. really experiment with and challenge the techniques to give guarantees about real systems. Admittedly, this is a hard task.

At a later stage when more applications have been studied (and the formalisms are perhaps more advanced) it would be interesting to try to estimate how well they can be used for simulation, but for now we restrict ourselves to the evaluation above.

We proceed to treat each piece of related work in turn, and then we draw conclusions based on the treatment.

6.1.2 Context calculi versus process calculi

In [RJP04] the difference between context calculi and process calculi is considered to be the presence of constructs to explicitly model context interaction. Another way to express the difference is given in [Bra03], where process calculi are described as formal theories of concurrent, distributed systems taking advantage of algebraic

reasoning, and context calculi should separate process behaviour from the (multiple notions of) computational context. This is our criterion for picking related calculi. This rules out traditional process calculi such as the π-calculus.

6.2 Context UNITY

In this section we treat the Context UNITY framework of Roman, Julien, and Payton [RJP04, JPR04].

6.2.1 Report

Context UNITY is a specialisation of Mobile UNITY [RM02, RMP97] to provide constructs to reason about interaction with the context. There are two goals; (1) to simplify development of context-aware applications, and (2) to gain a better understanding of the essential features of the context-aware computing paradigm. Context UNITY programs can be translated into Mobile UNITY programs (with the exception of non-deterministic assignment), which means that Context UNITY can largely be considered a syntactically sugared version of Mobile UNITY. Underlying the Mobile UNITY syntax is a translation into a first-order Hoare-style temporal logic [RM02].

The next two subsubsections (6.2.1.2 and 6.2.1.3) give a quite detailed walkthrough of Mobile UNITY and the additions to obtain Context UNITY. We have devoted ample space for this treatment, but the reader may be satisfied with the resume given first.

6.2.1.1 Context UNITY, very briefly

"Context UNITY represents an application as a community of interacting agents." [RJP04]. Each agent is uniquely identified and its behaviour exclusively defined by a program describing its interaction with variables. An agent interacts with its context by reading and writing (actuation) special variables and can itself decide which parts of the context that it finds interesting. The variables are governed by agent-specified (guarded) rules, thus separating the management of an agent's context from its internal behaviour. The unprecitability of context-aware systems is implemented by non-deterministic assignment statements. Agents have a location implemented by a special variable, which can be manipulated by the agent itself (subjective movement) or by the operational environment (objective movement). Agents run concurrently. A simple museum guide system was briefly sketched in [RJP04].

6.2.1.2 Mobile UNITY

We review Mobile UNITY as a basis for Context UNITY highlighting some important features to give the reader an intuition of the approach. Mobile UNITY captures the notion of location and movement across logical spaces while providing

formal reasoning via assertions. To this end the formalism has a notation for expressing mobile computations and a logic for reasoning about their temporal properties. Thus, Mobile UNITY can be said to extend the UNITY [CM88] model of concurrent computation by adding constructs for component location and transient interactions among components. Mobile UNITY aims to decouple a program's internal functionality from its interaction with the computational context.

Here is an overview, which will be explained in more depth afterward, of a Mobile UNITY program:

- A *system* consists of *program declarations*, a *components* section, and an *interactions* section.
- A program has a *location*, declares *local variables* with their initial values, and specifies clauses to control subsequent assignment to these variables. A clause can be the asynchronous conditional *when* or the synchronous conditional *reacts-to*. Assignments can also be enclosed in an *inhibit-when* statement, which disallows assignment under certain conditions.
- The components section instantiates the programs.
- The interactions section defines how the programs can interact because it has access to the variables of the programs.

In Mobile UNITY one specifies a *system* consisting of *programs* (processes) running in parallel, non-deterministically scheduled in a weakly-fair manner. The key elements of program specification are variables and (labelled) conditional multiple assignment statements. Programs are sets of conditional assignment statements, and each program has a location, which is a variable outside the Mobile UNITYmodel (and thus parametrises the model). Programs have *declare* and *initially* sections much like imperative programs. The *assign* section defines a program's behaviour. The transient interactions in Mobile UNITY consist of four additions to UNITY: *Transactions* (sequential critical regions), globally unique *labels* (on statements), *inhibitors* (strengthening of guards by predicates), and *reactive statements* which are to be executed to fixed-point, interleaved, after any other executed statement including those in transactions. One observes that the reactively augmented statements make up the basic atomic state transitions of the Mobile UNITY model. The *components* section defines which programs that make up the system and their initial locations. The *interactions* section defines interaction between the programs. Programs interact solely via shared variables. Regarding location it is worth remarking that programs have subjective (or local) movement, i.e. that can access their own location variable. This can, e.g., be used to enforce co-location during process communication. Shared variables are transient in the sense that sharing is controlled by a predicate (*when*) guarding the assignment. Comparing reactive statements and when-conditions, one one can think of reactive statements as providing context information in an eager manner, while when-conditions are lazy. The logic is used for reasoning, e.g. about

safety and liveness properties of the system, see [RM02] for a formal definition of these properties.

As mentioned in [RM02], the Mobile UNITY notation is very expressive, but should be restricted in practice to obtain, e.g., termination guarantees and more efficient implementation.

6.2.1.3 From Mobile UNITY to Context UNITY

In [RJP04] it is stated that context models should have the following properties:

- Expansive: The scope of context of a particular agent should not have a priori limits.
- Specific: It must be possible to specify tailor-made context definitions for an agent – context definitions should be modifiable.
- Explicit: Agents control their contexts – this requires an explicit notion of context.
- Separable: An agent's context specification can be separated from its behaviour specification.
- Transparent: The agent should be freed from the operational details of discovering its own context.

In Context UNITY context is defined from the perspective of a single component. One can think of a component (agent) as a state transition system; state change (representing change in context) occurs spontaneously without agent control (cf. the weakly-fair scheduling). Behaviour (and state) of a program is defined exclusively through its interaction with variables. There are three important variable types. *Internal*: These are not accessible/visible outside the agent. *Exposed*: Public variables that can be part of other agents' context. (These have access control associated with them.) *Context*: Variables that directly model, access, and modify context in a program by getting and putting information from/to exposed variables of other agents according to the agent-specific *context rules* (see below). In addition to context variables a program now has a *context section*, which provides rules to manage an agent's interaction with its desired context., i.e sensing of information from the operational environment[1], and affecting other agents by impacting their exposed variables. The context section thus secures decoupling of an agent's internal behaviour and management of its context; it is said to allow *projections*, i.e. local change can imply global change. An agent can also feed back information into its own context. We take a short digression into exposed variables before continuing with non-deterministic assignment. Exposed variables have an in-built access control policy; exposed variables consist of six components, one being a function from the reference agent's[2]

[1] In Context UNITY, the operational environment consists of everything that could potentially effect a program (computational agent), and the part that does is named context.

[2] By reference agent we mean the agent whose context we are considering [JR05].

credentials (see below) to a set of operations on that variable (e.g. read or write). Four such exposed variables are built-in: Location, type, agent identifier, and credentials. The type variable holds the program's name, while the agent identifier is the unique global id of the agent. In the credentials variable a profile of attributes for the program is stored for querying by other agents regarding access to their exposed variables. Returning to non-deterministic assignment; to handle the lack of a priori knowledge about the operational environment, Context UNITY introduces non-deterministic assignment statements to be used in context rules. A context rule governs the interactions associated with a certain context variable. The rule can quantify over variables that are place-holders for other agents' exposed variables (via the *uses* construction), and define restrictions (via the *given* construction) and interactions (via the *where-becomes* construction) with these. Context rules can be declared *reactive*. Systems in Context UNITY also have a *governance* section, which contains rules to capture behaviours that have universal impact across the system. This happens through assignment to exposed variables, and governance rules are assumed to be safe (and thus do not involve credentials). Objective (so-called global) movement can, e.g., be implemented via governance rules.

6.2.1.4 Formal differences

Formally, there is only one difference between Mobile UNITY and Context UNITY, namely an additional proof rule for non-deterministic assignment to context variables modelling the unpredictable real world. It is claimed that the other constructs can be translated into Mobile UNITY constructs [RJP04]. This sounds plausible.

6.2.1.5 Middleware support

Two middlewares have been used in conjunction with Context UNITY; EgoSpaces and LIME.

EgoSpaces [JR02, JR05] is a middleware that provides context information to applications in an abstract form. EgoSpaces evolved from LIME. As the authors write in [JR05]: "LIME requires strong assumptions about the operating environment that fail to hold as the number of devices, connections, and the degree of mobility grows." EgoSpaces is claimed to overcome these weaknesses. EgoSpaces is compared to the Context Toolkit in [JR05], and there EgoSpaces is found to be more suitable for supporting development of context-aware applications (in an ad hoc network scenario) because it addresses an application's need to dynamically discover and operate over a constantly changing context. An important notion in EgoSpaces is a *view*. A view is a projection of all data available to the reference agent. Views can be created, redefined, and deleted, and are defined over network, host, agent, and data constraints. Views are updated when agents access them, and thus provide asymmetric coordination. EgoSpaces provides triggered reactions and also "migrate", "duplicate", and "event" primitives, since these have been found common and useful in practice. EgoSpaces is implemented in Java using tuple spaces (via Elights), the CONSUL framework is used to collect context information from sensors and other

agents close by, and the SICC protocol for making a network tree and sending messages. There is, surprisingly, no discussion of whether such a location tree suffices for these purposes, we would suspect not. Performance simulation has been done via the OMNet++ discrete event simulator.

The LIME [MPR06] coordination middleware is an implementation of a LINDA-like tuple space calculus supporting mobility, and it has been specified using Mobile UNITY, and also in CRSs (see below). To demonstrate how LIME can be used as a context-aware middleware supporting context-aware applications an example, namely an application for tracking users via GPS called TULING [MP04], was modelled. A lightweight version of LIME is Limone [FcRH04], which "...centers the coordination tasks around *acquaintances*, and knowledge of specific coordinating partners is essential to Limone's functionality." [JR05]. Thus, it fails to capture the unpredictability of ad hoc networks.

Numerous other middleware systems exist, but we refer to [MP04] for an overview of these, since middleware as such is not the focus of this report. We do, however, mention Klaim, which is a formalism (process calculus) to support computing with mobile processes and explicit localities. It uses LINDA-like tuple spaces. Furthermore, a modal logic has been developed. There is also a programming language X-Klaim, based on Klaim, for programming distributed applications with mobile code. A compiler from X-Klaim into Java using the Klava package (run-time system for Klaim).

6.2.1.6 The Context UNITY logic

As mentioned, the logic underlying Context UNITY is a first-order Hoare-style modal logic. Properties are proven by constructing proof trees.

6.2.2 Evaluation

Negative context information is represented by using inhibitors, i.e. guards on context variables. Thus, rules only fire when the guards allow it.

There are different types of variables and guarded assignments to these. We also have transactions, inhibitors, context rules, credentials et cetera. Context UNITY is convenient for programming.

All agents are "on the same level" topologically, like in CRSs. Thus, it is not clear how one would represent a hierarchical location topology.

If we implement a graph-based topology, then symbolic range queries are unsuppported.

Probably, agents would interact with the topology via context variables. An agent could then inform other agents of its movements.

The modelling effort may not be low because of the location topology problem, but Context UNITY is well-suited for many other context aspects: Context interaction (sensing and actuation) is sharply separated from internal program behaviour,

and is possible via context variables (in the context section). Non-deterministic assignment statements capture the a priori unpredictable operational environment.

To the best of our knowledge no location model has been formalised in Context UNITY, but we imagine that formally it would resemble the approach of [Leo98], because a first-order logic is used there also. However, some examples utilising location information have been encoded.

6.2.3 Reasoning in practise

To the best of our knowledge, the logic has not been used to reason about the examples implemented. All Context UNITY constructs can be translated into logic formulae, and it would then in principle be possible to build proof trees establishing certain properties. It does seem cumbersome, and formally handling and proving things about systems with non-determinism is notoriously hard.

This concludes our discussion of Context UNITY.

6.3 Contextual reactive systems (CRSs)

In this section we treat Braione and Picco's theory of CRSs [Bra03, BP04].

6.3.1 Report

CRSs is a generalisation of Leifer and Milner's RSs [LM00] in the sense that interactions between computational agents (processes) and the context are disciplined. In CRSs it is possible to specify the computational contexts under which a class of behaviours is allowed, and the contexts under which it is not. CRSs is a category theoretical approach inspired by that of reactive systems and bigraphs [LM00, JM04]. The motivation for CRSs is two-fold:

- To separate the process behaviours from the computational context.
- To allow the specifier to define the notion of context and the rules governing how it affects the processes.

The goal of CRSs is to devise a formalism for modelling real middleware, in particular LIME. It is claimed that process calculi have been successfully used for specifying the semantics of coordination models and languages, but that these do not sufficiently address the modelling of a changing computational context; many such calculi have a rather rigid built-in context notion, i.e. a tight coupling between context and process. We agree.

In CRSs computational steps are described as transitions that rewrite terms and thus change the state of the system. The trouble with RSs, in the scenario of context-awareness, is that every computational context may host any interaction which makes it difficult to represent dynamic, subjective behaviour. We think of reactive systems as specifying the *internal* semantics of a system. In CRSs, it is possible to specify the

computational context under which a class of interactions is allowed (via *enablers*), and under which it is not (via *inhibitors*). Context is a first-class element of the formalism. We recite the relevant definitions (Def. 1, 2, 3, and 4) of [BP04] to make the following discussion more clear.

Definition 6.3.1 (Reactive System). *A* reactive system *(RS) is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *quadruple, where* $\mathcal{C}$ *is a category,* $I \in Ob\mathcal{C}$, $\mathbf{R} \subseteq \cup_{x \in Ob\mathcal{C}} \mathcal{C}(I,x) \times \mathcal{C}(I,x)$, *and* $\mathcal{D} \leq \mathcal{C}$ *is composition-reflecting, i.e.,* $D_0 D_1 \in Mo\mathcal{D} \implies D_i \in Mo\mathcal{D}$, $i = 0, 1$.

The morphisms of $\mathcal{C}$ are contexts ("terms with a hole"), and *ground* contexts are processes ("terms where the hole is replaced by a compatible sub-term", by morphism composition), denoted by $\mathcal{C}(I, x)$. Notice that contexts have exactly one hole. $\mathcal{D}$ specifies *reactive contexts*, i.e. contexts under which a rule may fire, and $\mathbf{R}$ is the set of *elementary* rewrite rules on processes; rewrite rules are pairs (l, r) of ground contexts, where l is named *redex* and r *contractum*. Extending (composing) elementary rules with reactive contexts, along with the following reaction relationship, yields *composite* rules:

Definition 6.3.2 (Reaction relationship). *The* reaction relationship, $\rightarrow$, *is defined as follows:* $a \rightarrow a' \iff \exists (l, r) \in \mathbf{R},\ D \in Mo\mathcal{D}\ .\ a = Dl \ \wedge\ a' = Dr$.

This relationship contains both elementary and composite rules of the RS so $\mathbf{R} \subseteq \rightarrow \subseteq \cup_{x \in Ob\mathcal{C}} \mathcal{C}(I,x) \times \mathcal{C}(I,x)$. A motivating example of context-aware printing is given and it is remarked that "it is not possible to forbid the reduction of a redex based on the properties of the context it is immersed in" [BP04]. (It is this example that we encoded in chapter 3.) To alleviate this, CRSs are proposed which allow elementary rules to be extended only be some instead of any active contexts. The following definition captures exactly this:

Definition 6.3.3 (Contextual reactive System). *A* contextual reactive system *(CRS) is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D}, \mathcal{D}[\![l, r]\!])$ *quintuple, such that* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *is a RS, and* $\mathcal{D}[\![l, r]\!]$ *is a function mapping any elementary rule* $(l, r) \in \mathbf{R}$ *to a composition-reflecting subcategory of* $\mathcal{D}$.

The reaction relationship is altered correspondingly:

Definition 6.3.4 (Reaction relationship for contextual reactive systems). *The* reaction relationship, $\rightarrow$, *is defined as follows:*
$a \rightarrow a' \iff \exists (l, r) \in \mathbf{R},\ D \in Mo\mathcal{D}[\![l, r]\!]\ .\ a = Dl \ \wedge\ a' = Dr$.

We see that different elementary rules may have different contextual constraints so some expressivity was gained. We remark that internal transitions performed by a group of processes still does *not* affect the surrounding context, i.e. there is no *actuation*.

Using *enablers* and *inhibitors*, for (elementary) rules, one can control what may and may not be present in the context for a rule to fire. Essentially, an enabler is a set, closed under morphism composition, which has as elements predicates on tuples. Such a set precedes a rule and guards applications of this, so to speak. Likewise for

inhibitors. We refer to [Bra03, BP04] for the formal definitions, as we here merely wish to pass on intuition. We do, however, recite an example from [BP04]. Here is a rule printing on a 'raw' printer, which does not fire if a PostScript printer is present, assuming a **print** primitive and a simple tuple space process calculus (cf. Table 1 of [BP04]):

$$\{- \mid \langle v \rangle \,.\, v \neq \text{pr:ps}\}^{\star} \quad \textbf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \longrightarrow P \mid \langle \text{job,txt,raw} \rangle \mid \langle \text{pr:raw} \rangle$$

where '$-$' denotes a *hole* in a context, and angle brackets denote tuples; $\langle v \rangle$ is a tuple with value v, for example. This rule can be rewritten using an inhibitor:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star} \quad \textbf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \longrightarrow P \mid \langle \text{job,txt,raw} \rangle \mid \langle \text{pr:raw} \rangle$$

The small c signifies that the tuples mentioned in the set must *not* be present in the context if the rule is to fire. Further, an enabler can be specified:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star} \quad \{- \mid \langle \text{pr:raw} \rangle\}^{\prime} \quad \textbf{print}(\text{txt}).P \longrightarrow P \mid \langle \text{job,txt,raw} \rangle$$

It looks simple, but is underpinned by a few technical contructions. What has happened in these two reformulation steps is essentially to move information from the term to the context of a rule.

6.3.1.1 Application of CRSs

CRSs have been used to formalise the core of the LIME middleware [Bra03], and also a tuple space process calculus based on Linda. The idea is to represent context by a global tuple space, thus separating specification of behaviour from specification of context where it may occur. In [MP04] there is a description of the TULING application which shows how location context can be made available in LIME. Thus, for location context information, LIME can be used much like a context toolkit, i.e. facilitate programming of mobile applications that need access to location information.

6.3.2 Evaluation

Negative context information can be represented by inhibitors.

CRSs can be thought of as a meta-calculus, like bigraphs. This has the advantage that different domain-specific calculi can be encoded. There are no in-built control structures to facilitate convenient programming. One can achieve control by representing calculi with control structures, but then the control structures depend on the calculus that is formulated as a CRS. As an example: For a process calculus based on Linda there will be operations for interacting with tuple spaces, and these will provide some control of computation.

Only flat location topologies can be represented, if represented as a term, and it is not feasible to have a hierarchical space reflected in the syntactic term structure in

the general case, as mentioned in [BP04]. This means that whichever calculus we wish to represent as a CRS we can only have flat terms.

Considering this limitation we can not support range queries in a natural way, nor nearest neighbour queries. Position and navigation queries do seem feasible, but we lack structure and this is likely to render programming of queries harder than in bigraphs and thus not convenient. As stated in [MP04], a query such as "find all components within a radius r from point (x, y)" can not be performed because that would require a range search inside the tuple space and LIME only provides value matching [MP04].

How interaction between a location-aware application a and the location topology t would be realised depends on the calculus represented as a CRS. No matter the choice of calculus, terms will be flat and a will live in the same system and at the same level as t.

The modelling effort would probably be high because we lack structure for our specific purpose despite encoding a suitable calculus.

6.3.3 Reasoning in practise

It is unknown to what extend the techniques of [LM00] can be used in the setting of CRSs, i.e. whether operational congruences can be derived automatically (from a RS to a LTS, i.e. from internal to external semantics). In [Bra03] it was necessary to restrict the definition of bisimulation to ensure that bisimilar processes have sufficient contexts in common. Furthermore, assessing whether adding negative context information increases expressiveness, needs to be explored. No logics exist for CRSs.

Collecting these facts it is fair to say that reasoning in practise is not feasible yet, and no such attempts have been made, to the best of our knowledge.

6.4 A calculus for context-awareness (CAC)

In this section we treat Zimmer's calculus for context-awareness (CAC) [Zim05].

6.4.1 Report

CAC "is a process calculus, whose aim is to describe dynamic systems composed of agents able to move and react differently depending on their location." [Zim05]. CAC features a hierarchical term structure like Mobile Ambients [CG00], and a generic *multi-agent synchronisation* mechanism inspired by the Distributed Join calculus [FGL+96], we remark that locations are organised in a tree. One can think of the calculus as a hybrid between the two calculi just mentioned, except that it also features non-local process synchronisation. The motivation is to develop a calculus that models how devices interact in a uniform way in wireless networks. Ambients are called *agents* and these represent locations, which are either physical or logical

units of computation. Agents have *definitions* that enable enclosed processes to perform reductions. The main feature of CAC is multi-agent synchronisation of tuples of values on *named channels*. Agents are not directly aware of their environment, but inform it of their capabilities by asynchronously sending *atoms*. The environment has definitions consisting of *rules* (think join patterns) to perform global synchronisation on these captured atoms; this is novel wrt. the Distributed Join calculus, where synchronisation happens locally and not across agent boundaries. There is a notion of priority (or scope) of patterns, namely that the deepest rule (pattern) matches first. Agents can move by the *go* primtive, but must give the explicit path to the destination because movement happens one step (up or down in the tree) at a time. There is no way to open or close agents, just like in Boxed Ambients [BCC01]. Definitions of a particular agent are activated (by a reaction rule) by adding them to the enclosing agent's definitions. Reaction rules rewrite an agent if an enclosed process matches one of the agent definitions (under some restrictions). We remark that contexts can have any number of holes, and that parallel composition in CAC is not commutative, which according to [Bra03] is unusual. We agree. The reason for this is that the names in the redex of a rewriting rule are bound pointwise in the process expressions on the reactum of the rule.

6.4.1.1 Example encodings and expressiveness

A small location-dependent printing example is given to illustrate how a process interacts with its enclosing agent. Further, a form of "remote procedure call" is encoded via continuations, and also a small packet routing protocol.

Expresiveness is investigated by encoding a monadic asynchronous π-calculus with replicated input, and a λ-calculus.

6.4.2 Evaluation

A basic model has been developed where agents may use different notions of computation on different physical locations.

In the current version of CAC it is not possible to express negative context information. The author suggests to add negated terms in pattern rules. This would be like adding "not-controls" (or co-controls) to bigraphs, which is inelegant because then there would have to always be one or the other. Introducing a notion of inhibitor like in CRSs or a sorting like in bigraphs is preferable.

The control structures in CAC are somewhat like those in bigraphs; a tree hierarchy of terms (agent processes), and a way to link agents – namely by channels and name restriction. Rules (patterns) are used to control reactions. These rules are part of the agents so contexts are not really separated from processes, at least not to the same extend as in CRSs or bigraphs. Anyhow, when programming with the calculus these structures and the movement primitive are useful. We emphasise the fact that each agent has its own set of rules, which can grow as other agents move into it so that it can activate their definitions. This is the way a changing context is modelled.

The location topology is a tree just like in Ambients. In bigraphs there is a forest of trees available.

Queries by an agent, for example a location-aware application, happen on the structure in which it resides. Probably, one would want to program an auxiliary process to traverse the tree and collect information. To do this the agent must know the topology of the entire tree because the *go* primitive needs an explicit path. This is incovenient and does not harmonise with the unpredictability of context in general, but is reasonable for a location hierarchy like a building. Apart from that, programming in CAC does resemble programming natively in bigraphs, and is thus likely to be equally inconvenient. In principle, it should be possible to support three of the four query classes discussed in chapter 2, but the nearest neighbour queries require some notion of distance.

Interaction is between processes in the tree structure.

The modelling effort regarding the hierarchy is low, but high wrt. queries. We find that encodings of larger examples are required to further test the modelling capabilities of the calculus.

6.4.3 Reasoning in practise

More work is needed wrt. the behavioural and equational theory of CAC. Thus, the calculus is not yet ripe for reasoning about systems in practise.

6.5 A formal model for context-awareness (CONAWA)

In this section we treat Baun Kjærgaard and Bunde-Pedersen's effort to define a formal model for context-awareness [KBP06a, KBP06b].

6.5.1 Report

Like other approaches, [KBP06a, KBP06b] argue that we lack formal support for realistic context-awareness. Furthermore, it is claimed that existing calculi [BP04, RJP04] only deal with very limited notions of context, and that a flat space structure does not suffice while being difficult to navigate. Also, context is not just physical location, but also logical information.

The approach taken in CONAWA takes origin in the Ambient calculus, but instead of having one tree representing space it has several so-called *views*, much like the place graph of bigraphs. The intention is to have one tree (view) for each category of context information needed by the application, e.g. locations and printer types.

In CONAWA, ambients are divided into two syntactic classes: *Context* and *reference* ambients. Context ambients (views) have unique names, are static in the sense that they can not move (navigate views), and can only be created (declared initially) and opened (a standard capability). Reference ambients are embedded in context

ambients and navigate these by exercising the capabilities *in*, *out*, *enter*, *exit*, *coenter*, *coexit*, and *open*. *in* and *out* are not observable by the context, whereas *enter* and *exit* are. Using the co-capabilities requires the context to allow these movements. Capabilities are instrumented with two pieces of information; a boolean expression over contexts to define which contexts the owner of the capability can be performed wrt., and the names of the reference ambients the owner of the capability can exercise the capability on. A wildcard name is included to match all reference ambients. Reference ambients can be replicated and can input/output names locally. A reference ambient will have a single (for consistency) presence in one ore more views simultaneously by a reference, e.g. a printer ambient has a type and a location. The capabilities *in* and *out* of the Ambient calculus are proposed extended to enable navigation in several views at once. A reference ambient navigates views by explicitly giving the path to collect context information. This reflects the idea that computation is seen as embedded in a number of contexts at the same time. Communication is local, i.e. an ambient (or rather a reference to an ambient) may communicate with other ambients which are its siblings or father in the tree where it currently resides. They communicate through the *ether* of each view, much like a tuple space, with no channels involved. Actions and capabilities are restricted by boolean expressions over contexts. Name scoping and general output paths have been left out of the calculus for simplicity. This model is parametrised over the notion of "proximity".

6.5.2 Evaluation

The authors evaluate their calculus by modelling examples of the four types of context-aware applications described in [SAW94], to which we return in chapter 7. For now, it is enough to know that the four application types are categorised from a user interface (UI) parspective according to whether they provide information or supply commands, and whether they are invoked manually or as a reaction to the current context. The authors find that representatives (in the domain of "pervasive health-care") of all four types could be modelled in CONAWA. Two thematic examples are "find the nearest available doctor" and "update a context/view using a reference process".

Contexts are special uniquely named ambients. Reference ambients may move around in the contexts by exercising capabilities. These capabilities may be instrumented with a "boolean" expression stating which contexts they match and which they do not, i.e. in which contexts the capability can be used. This is decided by naming the matching contexts, and putting a negation sign in front of the names of non-matching contexts. This is not the same type of negative information as for example the inhibitors of CRSs. The difference is that inhibitors limit reaction to certain contexts where something is not present, but "boolean" expressions discriminate named contexts and not their contents. This is a gain of introducing views. We also have views in bigraphs in that a place graph is a forest of trees that can be made uniquely identified by requiring each tree to have a unique control on the root node.

There are some important control structures; views, (guarded) capabilities, and

agent references. Agent references allow the specifier to refer to an agent to partake in different contexts, i.e. different views on the world/situation, simultaneously. The guards can be used to control in which views a given capability can be exercised, which resembles programming with conditionals.

The location topology is a forest of uniquely identified trees, i.e. a collection of views.

The only query that lacks support is "nearest neighbour". You have to consider the entire system to formulate single ambients. Specifically, an agent has to explicitly give a path for moving, which may be an unreasonable assumption because it requires detailed a priori knowledge of the whole operational environment and not just the context at hand. This does not reflect the ad hoc nature of the real world, but is reasonable enough for a location hierarchy.

Interaction between a location-aware application a and the location topology t can happen by programming a to traverse t. An idea is to have an auxiliary agent collect this information. The authors suggest to introduce designated reference agents to update contexts, i.e. to act as carriers of sensor information. An example is given in the discussion of [KBP06a], which requires the ability for reference agents to output capabilities, and not just names. Having reference agents invoke these "sensor agents" could be considered actuation. Still, there is no *representation* of the world (like **C** in Plato-graphical systems).

We remark that reference ambients can not remove themselves from views, so a device will always have some location once it has been located once. This is not a problem for location models, as seen in chapter 5.

The modelling effort is probably on level with that of CAC since both are Ambient-based process calculi, albeit with some differences (patterns versus guarded capabilities and views).

6.5.3 Reasoning in practise

No behavioural or equational theory has been established for the calculus, nor any expressivity results. There is no formal semantics of the calculus, merely a few examples of what reduction rules could look like.

We conclude that much work is needed before any formal reasoning can be carried out.

Here are some suggestions for corrections:

- Considering Table 1 of [KBP06a], presenting the syntax of CONAWA, it can be seen that there is no base case for the syntactic categories **C** and **R**, thus the inductive definition is not well-founded.
- The important example in Figure 11 is not syntactically correct because (1) a reference agent 'FNDAP2' is (illegally, see Table 1) used as prefix to a capability, and/or (2) the square brackets do not match.

It should, however, be possible to correct these errors.

6.6 Other approaches

We mention a work where a location model is formalised in first-order logic, two works in progress, and one piece of research formalising context from the viewpoint of artificial intelligence (AI).

- A formal location model in Z/Eves [Leo98].
- The Agent Distributed π-calculus (AgDpi) [Hen05, Hen04].
- The N$^\sharp$ programming language effort [WBB06].
- "Formalizing Context" [MB97].

In appendix C of [Leo98] three location services are specified in the Z formalism, which is a first-order logic [MS97]. Here, we merely give the reader a taste of the approach. A service consists of a location hierarchy, updates on this hierarchy, and some queries. We briefly consider the symbolic one. First, two object types are declared; LOCATION and OBJECT. A location hierarchy is then declared as an assymmetric and transitive inclusion ordering. Predicates in locations corresponding to the relevant spatial relationships are also defined. Here is a parametrised location query which for all located-objects at a given location:

$$target? : LOCATION \land result! : \mathcal{P}(OBJECT) \implies$$
$$result! = \{x : OBJECT \mid (x, target?) \in locatedAt\}$$

where 'locatedAt' is a predicate which decides whether a given object is in a given location, and $\mathcal{P}$ is the power set. A sighting operations is also defined, along with some other queries. Thus, set theory is used as a programming language.

AgDpi is Distributed π-calculus (Dpi) with *nominal agents*. In AgDpi there is mobile code running inside nominal agents. Dependent types are used to enforce selective access (read/write capabilities) to resources. Locations are unique and organised in a flat structure. Communication is local and authenticated (via types). A special kind of channel *disc* is used by agents to discover local resources, and then the agents act accordingly. This work is worth following should it progress from the current "work in progress" status, e.g. emerge as a full-fledged process calculus.

In [WBB06] a first step is taken toward a programming language for pervasive applications based in the Ambient calculus. The language is called N$^\sharp$ because its syntax resembles C$^\sharp$ or Java. Communication is between ambients and processes is asynchronous. Named ports are adopted from the π-calculus to facilitate easy message passing. A prototype compiler exists.

In [MB97] context is formalised as a first-class object, and can be thought of as a generalisation of a collection of assumptions (in a Gentzen style logic). A context may even correspond to an infinite and only partially known collection of assumptions. The point of origin is artificial intelligence, and the formalism used is a first-order logic. There is no clear relation between this work and our field of research so we refrain from further discussion of this work.

6.7 Concluding remarks

We structure our remarks according to the method above.

6.7.1 Evaluations

Negative context information, e.g. in the form of inhibitors is certainly a useful feature. Whether it proves necessary for modelling ubiquitous systems in bigraphs and Plato-graphical models is uncertain. We suggest more modelling experience for deciding this.

Control structures decide how convenient the programming task is. In the calculi where hardly any are available, it seems unrealistic to model and program realistic systems.

Hierarchical location topologies should feature in calculi for location-awareness. Considering how other aspects of context may very well be hierarchical in nature, e.g. the organisation of a company, we conjecture that flat topologies are not sufficient.

Like for location, queries on the context are best supported if the context is structured. Furthermore, control structures help.

Interaction between a location-aware application and the context topology can become complicated if the application itself is part of the topology. Separating concers, as in Plato-graphical models, is useful.

The modelling effort is high when programming directly in meta-calculi. One can gain control structures by encoding other more domain-specific calculi though. As far as we know, it is a novelty to explicitly represent the world as a system in its own right, as done in Plato-graphical models. It is this feature that is the basis of our simulation idea.

6.7.2 Reasoning in practise

None of the works considered here have been used for reasoning in practise. Nor have bigraphs or Plato-graphical models. Tool support seems to be required to really make progress in this area.

6.7.3 Summa summarum

On a high level we can say that further experimentation with large examples is needed, and that tool support is essential in this effort. Much work in improving the theories tools also persists.

Chapter 7

Future Work

7.1 Possible directions for future work

We have identified some directions for future work.

- Characterise context-awareness in terms of Plato-graphical models and enrich our model to support this.
- Model a real-life system.
- Create a list of properties one wishes (to prove/guarantee) for context-aware systems.
- Work on a tool for BRSs to enable experimentation with models formulated in bigraphs, and simulation of systems – to really test how useful Plato-graphical models are. (The first question of section 1.3.)
- Investigate formal reasoning about Plato-graphical systems, perhaps by studying a form of bisimulation between BRSs. (The second question of section 1.3.)
- Formally state and prove a dynamic correspondence between Ξ programs and their bigraphical images under $[\![\cdot]\!]_X$.
- Enhance bigraph theory.

We discuss each one in turn.

7.1.1 Characterising context-awareness

Motivated by the fact that the notion of context is still ill-defined [DA00], we strive for a finer taxonomy of context with the purpose of a "context checklist" for applications, which could help to define needed components in a library for context-aware

programming. We believe that characterising the context types of [SAW94] in terms of Plato-graphical models will aid in understanding of context-awareness as such by sharpening the definitions. By formalising an application's context interaction as interaction between Plato-graphical components we gain precision.

We briefly recall the four types of context-aware applications that are mentioned in [SAW94]. The types are along two axes: Manual vs. automatic, and information vs. command. Manual and automatic refer to whether the user has to do something to make the application either fetch information or perform an action specified by the command. *Proximate selection* has to with finding or emphasising the located-objects that are nearby, and is a manual information task. *Automatic contextual reconfiguration* is an automatic information task that adds or removes components (typically software) or alters connections (typically wireless) depending on the context. *Contextual information and commands* are commands whose execution depend on the context – printing to the nearest printer will have a different result depending on the user's location. *Context-triggered actions* are simple 'if-then' rules used to specify how context-aware systems should adapt, and context-triggered are invoked automatically according to these rules. This is enough knowledge for our purposes.

Now, consider the following setup (suggested by Niss) depicted in figure 7.1.

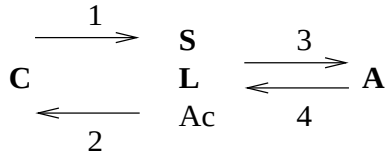


Figure 7.1: The four categories of context-aware applications as Plato-graphical interaction.

We imagine the following four interactions:

1. The proxy **P** possesses a sensor **S** which senses reconfigurations in the context **C** and informs the model **L**.
2. **P** is extended with an *actuator component* (Ac) which can affect **C** on behalf of **A**, i.e. make it reconfigure.
3. The agent **A** is informed of relevant context change by **P** (**L**).
4. **A** affects **L**, i.e. makes it change its conception of the context information.

One can think of (2) as actuation, e.g. if the agent wished to turn on the light in a dark room. (4) represents the ability to override the model if it, e.g. has an inaccurate or wrong conception of the context. (3) can be either "manual" or "automatic" (to use the terms of [SAW94]), with the manual case being the agent asking for information, and the automatic case being some sort of event or call-back. (1) can likewise be divided into manual and automatic.

We claim that this setup generalises [SAW94]. "Automatic contextual reconfigurations" are handled entirely in **C**. "Context-triggered actions" is the call-back of (3) mentioned above. "Proximate selection and contextual information" is the manual version of (3). "Contextual commands" are more complicated. We think of this as a sequence of interactions; first **A** asks **P** for the relevant context information and then it issues a command by (2) and (4) above.

Once established the characterisation should be challenged by capturing other informal characterisations of context-aware interactions. In [Sch95] the following questions for determining *situations* are emphasised: Where the user is, who the user is with, and what resources are nearby. The components are device agents (that maintain status and capabilities of devices), user agents (that maintain user preferences), and active maps (that maintain location information of devices and users). Another work to draw challenges from is [DA00] where the computing environment consists of CPUs, devices, and network connectivity. There is also a notion of user environment characterised by location and nearby people. Furthermore, the physical environment such as lighting and noise level is important. The model requirements here include interpretation, acquisition, and storage (history) of context.

This characterisation may serve as a framework for comparing concrete context-aware models (of realistic systems).

7.1.2 Formalising realistic examples

We have identified the following directions:

- Model the whole of ITU.
- Model a real-life system.
- Model a protocol for delivering messages in MANETs such as Geocast [DR03].

Modelling ITU should be straightforward. Picking a suitable real-life system is not easy, and requires more consideration. Modelling a protocol like Geocast seems to require devices to contain messages, and some sort of reachability information inherent in the topology.

7.1.3 A list of properties

We should create a list of properties we want (to prove/guarantee) for context-aware systems. All properties should be relevant for real-life systems and also be provable by the reasoning principles available. Some properties may not be provable with the current techniques so they may give rise to research of new reasoning principles for bigraphs or Plato-graphical models.

7.1.4 Tool support

In the bigraphical Programming Languages group at the IT University of Copenhagen, some members are currently working on a prototype implementation of BRSs. Normalisation of binding bigraphs has been implemented. The next step is to implement an algorithm for matching (like the examples we saw in chapter 4) [BDGM06]. Perhaps, it is an idea to implement an add-on for *local* bigraphs. This tool will be crucial for simulation purposes because realistic examples easily become too large to handle manually, let alone reason about. We need tool support to truly conduct experiments with real systems. We have already defined a translation from Ξ_{sugar} into the implementation of binding bigraph terms. Assuming that we can implement a simulator it will become relevant to look a location event generation – perhaps inspired by the generic location event simulator of [SC02].

Despite not having written much about this direction for future work, we believe it to be crucial for answering the first of the two open questions of section 1.3, in part because it supports the preceding three directions, and also reasoning *in practise*.

7.1.5 Formal reasoning

Proving properties about the bigraphical location model could be desirable, but it is unclear which properties we wish to prove and with which techniques. Certainly, one possible direction is to make precise some criteria for when components of a Platographical system can be substituted (while maintaining properties of the system as a whole). This has to do with bisimilarities between BRSs.

We may wish to prove properties such as, e.g., access control. This may involve using BiLogics [CMS05], or perhaps access control could be ensured via sorting (only allowing devices with a particular access token to enter a room with a matching token), and then proving that the system can not place an illegal device inside a protected room, by rule induction. Further studies in desirable properties of context-aware systems are needed. We have uniqueness of device locality by invariant (rule induction).

A technique for securing certain properties of our models could be to use sortings further. We believe that Søren Debois is currently working on transferring the work of [DBH06] from the setting of RSs to bigraphs. A clever thing about this, from our viewpoint, is that it enables us to express negative context information, by removing the unwanted bigraphs from the system. We could perhaps also impose a building sorting to ensure that certain locations (perhaps identified by a internal type control) are not within certain other locations, e.g. we do not wish for buildings to be within rooms. To this end we envision using a *predicate sorting* as defined in (Definition 15) of [DBH06]. In [DBH06] a sorting of a category $\mathcal{B}$ is a functor into $\mathcal{B}$ that is faithful and surjective on objects (Definition 4 of [DBH06]). This allows us to define *decomposible* predicates to filter out unwanted morphisms (bigraphs). To combine sortings, Debois, intuitively, combines predicate sortings via conjunction by a pullback construction (Proposition 4 of [DBH06]), whereas Høgh Jensen

composes sorting functors. Thus, we can combine an additional sorting with the Plato-graphical sorting.

7.1.6 Dynamic correspondence

Ξ-programs and their images under $[\![\cdot]\!]_X$ evaluate in one-to-many correspondence, i.e. the bigraphical representation takes one or more steps for each Ξ reduction step. Thus, we would like to prove *something like* the following "conjecture".

Conjecture 7.1.1.
$\forall e, \sigma, X, s, g.\ (\exists\, e', \sigma'.\ \langle e, \sigma\rangle \to \langle e', \sigma'\rangle \ \wedge\ [\![\langle e', \sigma'\rangle]\!]_X = /\vec{Y}\ .\ g \mid s) \implies$
$([\![\langle e, \sigma\rangle]\!]_X \longrightarrow\!\!\!\!\!\triangleright^{+} /\vec{Y}\ .\ g \mid s)$, *where* $fv(e) \cup fv(e') \subseteq Y \subseteq X$, *and* $\longrightarrow\!\!\!\!\!\triangleright^{+}$ *is the transitive closure of* $\longrightarrow\!\!\!\!\!\triangleright$.

The idea of a proof should be: Analyse each of the cases of possible reaction in Ξ. We need a substitution lemma for the cases where evaluation results in a substitution. If we can prove such a lemma, then the result can be lifted to evaluation contexts by the lemmas 7.1.1 and 7.1.2. However, a substitution lemma seems difficult to prove because we may have to perform a substitution on a lambda abstraction, which is a *passive* control in bigraphs! It seams that we need some sort of logical relation to resolve this.

Lemma 7.1.1. *For any evaluation context E, term e, and set X such that $fv(E[e]) \subseteq X$, it holds that $[\![E[e]]\!]_X = [\![E]\!]_X \circ [\![e]\!]_X$.*

Proof. The proof should be by structural induction on E.

Lemma 7.1.2. *This lemma should correspond to definition 4.3.1.*

A proof should be by structural induction on evaluation contexts.

Conjecture 7.1.2. *We conjecture that "the opposite direction" of Conjecture 7.1.1 can be proven, but it seems that we need some way to keep track of the bigraphs that* are *images of Ξ-terms.*

We emphasise the fact that the "conjecture" does not hold as stated, but is merely stated to give the reader an idea of what we would like to prove. The reason that it does not hold is that we can not perform substitution under passive controls. However, this does *not* make the encoding wrong, because when such a control, e.g. a lambda abstraction, is applied eventually, it will become possible to perform the substitution and thus "catch up".

One could also consider studying other programming language issues in the setting of bigraphs. It would probably be wise to do it in as simple a setting as possible. We do, however, not see this as important for our endeavours.

7.1.7 Enhancing bigraph theory

Currently, the following extensions are on the wish list:

- DAGs: Replace the place graph (a forest of trees) with a DAG. The motivation was given in chapter 5, where organising a building with rooms, wings etc. was a little troublesome.
- Time: Timed automata may inspire, we think of [D'A99].
- Stochastics: We think that some inspiration can be found in [Pri95, Bra02, D'A99, dAHJ01].
- Continuous space (hybrid systems): Possible works of inspiration (apart from the ones mentioned just above): [AD94, ACHH93, Hen96, DB96].

We discuss each item in turn in a little more detail after considering the overall purpose of them.

The visions mentioned in chapter 3 remain; DAGs, time, continuous space, and probabilistic information. Enriching the theory of bigraphs with these aspects is a demanding task, but certainly interesting.

7.1.7.1 DAGs

We decided to use the place graph for representing the location hierarchy instead of merely by linking, On first thought, it might seem reasonable to hierarchically order floors, wings, rooms, and devices in a building. Choosing one ordering has its drawbacks, however. Should wings or floors be higher in the tree? If we choose floors over wings then we could end up representing each wing on every floor, thereby introducing redundancy. This can, however, be remedied by using DAGs instead of trees. Furthermore, DAGs naturally support "shared locations" as, e.g., an auditorium residing on the floors simultaneously. If we shy away from altering the theory of bigraphs then DAGs could be implemented using several trees (roughly one for each "location sharing"), but navigating and keeping consistent several such *views* would complicate the modelling effort. In chapter 6 we discussed a piece of related work, where a sort of pointer is proposed to address this idea.

7.1.7.2 Time

A notion of time in the model is required to be able to order events, i.e. for example for $\mathbf{S_X}$ to be able to inform $\mathbf{L_X}$ of the order of sightings to facilitate a closer correspondence in the "states" c_X and l_X.

7.1.7.3 Probabilities

Probabilistic information is required to make the model more realistic, i.e. closer to the way real-life positioning systems work. This is desired for simulation purposes.

The probability information should be attached to events – perhaps as a tag along with a time stamp.

7.1.7.4 Continuous space

Continuous space has to do with geometric coordinates. It should be possible to determine the geometric whereabouts of located-objects, and to compute metric distances.

7.1.7.5 Summing up: Enhancements

To conclude, it seems relevant to consider these four aspects both from a theoretical and a practical point of view.

7.2 Concluding remarks

It seems that there are two roads to take.

- Continue on the experimental path.
- Delve into extensions of bigraph theory.

This concludes our suggestions for future work.

Chapter 8

Conclusions

In this report we have begun evaluation of our main hypothesis, which states that BRSs are suitable for programming, simulating, and reasoning rigorously about ubiquitous systems. To this end we have ascertained a more tangible hypothesis that supports the main hypothesis, namely that BRSs are suitable for modelling location-aware systems, specifically a sentient building. The supporting hypothesis has been ascertained by establishing the following:

- Based on an investigation of the literature on location models we have been able to come up with a representative example for our modelling effort; a sentient building.
- It is awkward to model context-aware (location-aware) systems directly in bigraphs.
- This awkwardness can be alleviated by using *Plato-graphical* models.
- Location models, of e.g. a sentient building, can be modelled and programmed conveniently as Plato-graphical models using a bigraphical encoding of a Mini-ML-like calculus with references. We have shown that the physical world and a simple positioning system can be conveniently model in bigraphs using reconfigurations. All this with an acceptable modelling effort.

The sentient building case study has helped us gain a better general understanding of which requirements context-aware (location-aware) programming puts on the theory used for the modelling.

This report has tried to take a step in bringing together the research on location models with the formal theory of bigraphs. It is clear that if theoreticians are to truly influence the way systems are built then we must try to work with the systems and problems that designers face.

Our study of related work shows that even though some theories and tools exist for supporting context-aware modelling and programming, there is still much

work to be done. The theories and tools do not capture time, continuous space, or stochastic behaviour. In addition, there are few frameworks supporting both theoretical modelling and actual tool support. To the best of our knowledge no simulators for context-aware systems exist, which we conjecture is an important challenge to overcome to push forward theory and technology.

We have identified several avenues for future research. One avenue, from our point of view, is to extend bigraph theory. Another avenue, which we intend to follow, is to expand the modelling effort to more realistic systems with support from a tool, which is to be developed simultaneously. The main goal is to be able to simulate real systems in bigraphs. A tool will also help to reason about real systems in practise.

We may be as bold as to say that our experimental endeavour has made research contributions, and hope that this piece of work can inspire researchers to work on realising the global ubiquitous computer. In the words of Professor Andy Hopper: "...I want to extend the computer and information systems to observe the real world and automatically modify the systems' behaviour to suit the prevailing conditions. Such "sentient" computing systems will play a key part in ensuring the sustainability of our planet." Quote from `http://www.cl.cam.ac.uk/Research/DTG/~ah12/aims-research.html`.

Chapter 9

Acknowledgements

Foremost, I wish to thank my lovely wife Dagmar and my parents Vibeke and Erik for being there for me always.

I would also like to thank the members of the Bigraphical Programming Languages group at the IT University of Copenhagen for constituting a very good research environment, and the following people in particular: Mikkel Bundgaard for being an excellent but pervasively bitter office mate, and for answering close to a plethora of my questions. Søren Debois for enlightening discussions (primarily about bigraphs), and his unique sense of humour. Troels Damgaard for conversations about bigraphs and life in general, and for making me realise when to say "no". My advisors Henning Niss and Lars Birkedal for working with me, and for their good advice during this piece of research.

A special thanks goes to my "psychologist" Jakob Grue Simonsen for being amenable to my feelings of inadequacy and general complaints. He certainly beats the Emacs Psychiatrist.

I also wish to thank my friends Jakob Lemvig and Jesper Rasmussen for helping me think about other things in life than work. Finally, I wish to thank Philip Bille for always showing off and making me want to become a better researcher.

Bibliography

[ACH+01] Mike Addlesee, Rupert W. Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, August 2001.

[ACHH93] Rajeev Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, 736, 1993. LNCS.

[AD94] Rajeev Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BBR02] Martin Bauer, Christian Becker, and Kurt Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. *Personal and Ubiquitous Computing*, 6(5-6):322–328, December 2002. ISSN:1617-4909.

[BCC01] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *In 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, volume 2215 of *LNCS*, pages 38–63. Springer-Verlag, 2001.

[BD05] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9:20–31, 2005. Springer-Verlag.

[BDE+05] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. Technical Report 74, IT University of Copenhagen, Rued Langgards Vej 7, DK-2300 Copenhagen V, November 2005. ISBN: 87-7949-110-3.

[BDE+06] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas T. Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of FoSSaCS*, volume 3921 of *LNCS*, pages 187–201, Vienna, Austria, March 2006. Springer-Verlag. ISBN: 3-540-33045-3.

[BDGM06] Lars Birkedal, Troels C. Damgaard, Arne J. Glenstrup, and Robin Milner. Matching of bigraphs. Technical Report ITU-TR-2006-88, IT University of Copenhagen, June 2006.

[BH06] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *In Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'05)*, volume 154 of *Electronic Notes in Theoretical Computer Science*, pages 7–29. Elsevier, 2006.

[BP04] Pietro Braione and Gian Pietro Picco. On calculi for context-aware coordination. In *Proceedings of COORDINATION'04*, volume 2949 of *LNCS*, pages 38–54. Springer-Verlag, 2004.

[Bra02] Mario Bravetti. *Specification and Analysis of Stochastic Real-Time Systems*. PhD thesis, Università di Bologna, Padova, Venezia, February 2002.

[Bra03] Pietro Braione. *On Calculi for Context-Aware Systems*. PhD thesis, Politecnico di Milano, Dipartimento di Electronica e Informazione, Piazza Leonardo da Vinci 32, 2003.

[BS01] Barry Brumitt and Steven Shafer. Topological world modeling using semantic spaces. In *UbiComp 2001 Workshop on Location Modeling for Ubiquitous Computing*, October 2001.

[BZD02] Michael Beigl, Tobias Zimmer, and Christian Decker. A location model for communicating and processing of context. *Personal and Ubiquitous Computing*, 6(5-6):341–357, 2002.

[CCK+05] Dan Chalmers, Jon Crowcroft, Marta Kwiatkowska, Robin Milner, Vladimiro Sassone, and Morris Sloman. Global ubiquitous computing: Design and science. Final draft, a newer version of the document can be found at `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/manifesto.pdf`, June 26 2005.

[CDMF00] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In *Proceedings of the 6th annual international conference on mobile computing and networking (Mobicom)*, pages 20–31, Boston, Massachusetts, August 2000.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report tr2000-381, Department of Computer Science, Dartmouth College, 2000.

[CM88] K. Mani Chandy and Jayadev Misra. Parallel program design: A foundation. Addison-Wesley, 1988.

[CMS05] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Spatial Logics for Bigraphs. In *Proceedings of ICALP'05*, volume 3580 of *LNCS*, pages 766–778. Springer-Verlag, 2005. ISBN: 3-540-27580-0.

[D'A99] Pedro R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.

[DA00] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness*, 2000. Part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000).

[dAHJ01] Luca de Alfaro, Thomas A. Henzinger, and Ranjit Jhala. Compositional methods for probabilistic systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, LNCS, pages 351–365. Springer-Verlag, 2001.

[DB96] Pedro R. D'Argenio and Ed Brinksma. A calculus for timed automata. In *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 110–129. Springer-Verlag, 1996. ISBN: 3-540-61648-9.

[DB05] Troels C. Damgaard and Lars Birkedal. Axiomatizing binding bigraphs (revised). Technical Report TR-2005-71, IT University of Copenhagen, 2005.

[DBH06] Søren Debois, Lars Birkedal, and Thomas Hildebrandt. Sortings for reactive systems. In Christel Baier and Holger Hermanns, editors, *In Proceedings of CONCUR*, 2006. To appear.

[DD05] Søren Debois and Troels C. Damgaard. Bigraphs by example. Technical Report TR-2005-61, IT University of Copenhagen, March 2005.

[DR03] Frank Dürr and Kurt Rothermel. On a location model for fine-grained geocast. In Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy, editors, *In Proceedings of UbiComp 2003: Ubiquitous Computing*, LNCS, pages 18–35, Seattle, WA, USA, October 12-15 2003. Springer Berlin. ISBN: 3-540-20301-X, ISSN: 0302-9743.

[DRD+00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(3):285–321, September 2000.

[FcRH04] Chen-Liang Fok, Gruia catalin Roman, and Gregory Hackmann. A lightweight coordination middleware for mobile computing. In Rocco De Nicola, Gian Luigi Ferrari, and Greg Meredith, editors, *Proceedings of the 6th International Conference on Coordination Models and Languages (Coordination 2004)*, LNCS, pages 135–151, Pisa, Italy, February 2004. Springer-Verlag.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (COCNUR)*, volume 1119 of *LNCS*, pages 406–421. Springer-Verlag, 1996. ISBN:3-540-61604-7.

[Har00] Robert Harper. Type systems for programming languages (draft). Notes, Spring 2000. http://www.cs.cmu.edu/ rwh/misc/tspl.pdf.

[HB01] Jeffrey Hightower and Gaetano Borriello. A survey and taxonomy of location systems for ubiquitous computing. Technical Report UW-CSE 01-08-03, University of Washington, Computer Science and Engineering, August 24 2001.

[HBB02] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello. The location stack: a layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE workshop on mobile computing systems and applications (WMCSA)*, pages 22–28, Callicoon, New York, June 2002.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[Hen04] Matthew Hennessy. Context-awareness: Models and analysis. Talk at 2nd UK-UbiNet Workshop, slides at `www.cogs.susx.ac.uk/users/matthewh/talks.html`, May 2004.

[Hen05] Matthew Hennessy. Towards a calculus for nominal mobile agents. Talk at Symposium on Trustworthy Global Computing in Edinburgh, slides at `http://www.cogs.susx.ac.uk/users/matthewh/talks/tgc05.pdf`, April 2005.

[HHS+02] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. *Wireless Networks*, 8:187–197, February 2002.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of Pervasive'02*, volume 2414 of *LNCS*, pages 167–180. Springer-Verlag, 2002.

[HNO06] Thomas Hildebrandt, Henning Niss, and Martin Olsen. Formalising business process execution with bigraphs and reactive XML. In Paolo Ciancarini and Herbert Wiklicky, editors, *In Proceedings of Coordination Models and Languages: 8th International Conference (COORDINATION 2006)*, volume 4038 of *LNCS*, pages 113–129, Bologna, Italy, June 14–16 2006. Springer-Verlag. ISBN: 3-540-34694-5.

[HNOW05] Thomas Hildebrandt, Henning Niss, Martin Olsen, and Jacob Winther. Distributed Reactive XML. In *In Proceedings of the 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord05)*, 2005.

[Hop00] Andy Hopper. Sentient computing? *Phil. Trans. R. Soc. Lond., A*, 358:2349–2358, August 2000. An abridged and updated version of the Royal Society Clifford Paterson Lecture 1999.

[Jen06] Ole Høgh Jensen. *Mobile Processes in Bigraphs*. PhD thesis, King's College, University of Cambridge, 2006.

[JM03] Ole Høgh Jensen and Robin Milner. Bigraphs and Transitions. In *Proceedings of POPL'03*, pages 38–49. ACM Press, 2003. ISBN 1-58113-628-5.

[JM04] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge – Computer Laboratory, February 2004. ISSN 1476-2986.

[JPR04] Christine Julien, Jamie Payton, and Gruia-Catalin Roman. Reasoning about context-awareness in the presence of mobility. In Antonio Brogi, Jean-Marie Jacquet, and Ernesto Pimentel, editors, *In Proceedings of the 2nd International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA)*, volume 97 of *Electronic Notes in Theoretical Computer Science*, pages 259–276, Marseille, France, September 2 2004.

[JR02] Christine Julien and Gruia-Catalin Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proceedings of the 10th International Symposium in the Foundations of Software Engineering*, pages 21–30, November 2002.

[JR05] Christine Julien and Gruia-Catalin Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. Technical Report TR-UTEDGE-2005-004, The University of Texas, 2005.

[JS02] Changhao Jiang and Peter Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 246–263. Springer-Verlag, 2002.

[KBP06a] Mikkel Baun Kjærgaard and Jonathan Bunde-Pedersen. A formal model for context-awareness. Report Series RS-06-2, BRICS, February 2006. ISSN: 0909-0878.

[KBP06b] Mikkel Baun Kjærgaard and Jonathan Bunde-Pedersen. Towards a formal model of context-awareness. In *Presented at the International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB). A Workshop of PERVASIVE 2006*, Dublin, Ireland, May 7th 2006.

[Leo98] Ulf Leonhardt. *Supporting Location-Awareness in Open Distributed Systems*. Ph.d. thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, May 1998.

[LM00] James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In Catuscia Palamidessi, editor, *Proceedings of CONCUR'00*, LNCS, pages 243–258. Springer-Verlag, 2000.

[MB97] John McCarthy and Saša Buvač. Formalizing context. In Atocha Aliseda, Rob van Glabbeek, and Dag Westersthl, editors, *Computing Natural Language: Working papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135. Stanford University, 1997.

[Mil02] Robin Milner. Computing in space, May 1 2002. A lecture by Robin Milner, for the opening of the Computer Laboratory's William Gates Building at the University of Cambridge.

[Mil04a] Robin Milner. Axioms for bigraphical structure. Technical Report UCAM-CL-TR-581, University of Cambridge – Computer Laboratory, February 2004. ISSN 1476-2986.

[Mil04b] Robin Milner. Bigraphs for Petri Nets. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *LNCS*, pages 686–701. Springer-Verlag, 2004.

[Mil04c] Robin Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, University of Cambridge – Computer Laboratory, September 2004. ISSN 1476-2986.

[Mil04d] Robin Milner. Local bigraphs, confluence and λ-calculus. DRAFT, October 29 2004.

[Mil05a] Robin Milner. Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 2005. To appear. Revised version of UCAM-CL-TR-581.

[Mil05b] Robin Milner. Bigraphs: A tutorial. Slides, April 2005. http://www.cl.cam.ac.uk/users/rm135/bigraphs-tutorial.pdf.

[Mil05c] Robin Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 2005. Submitted. Revision of UCAM-CL-TR-614.

[MP04] Amy L. Murphy and Gian Pietro Picco. Using coordination middleware for location-aware computing: A lime case study. In *In Proceedings of the 6th International Conference on Coordination Models and Languages (Coordination)*, Pisa, Italy, February 2004.

[MPR06] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering (TOSEM)*, pages 1–48, 2006.

[MS97] Irwin Meisels and Mark Saaltink. The Z/EVES reference manual (for version 1.5). Technical Report TR-97-5493-03d, ORA Canada, September 1997. http://www.ift.ulaval.ca/ jodesharnais/glo21941/ZEves/ZEvesRefMan.pdf.

[NGP05] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calculus for global computing. In *Proceedings of ICALP'05*, volume 3580 of *LNCS*, pages 1226–1238. Springer-Verlag, 2005.

[OJDA01] Thomas O'Connell, Peter Jensen, Anind K. Dey, and Gregory D. Abowd. Location in the aware home. In Michael Beigl, Phil Gray, and Daniel Salber, editors, *Location Modeling for Ubiquitous Computing – UbiComp 2001*, Atlanta, Georgia, September 30 2001.

[PlaBC] Plato. The republic, book vii, 360 B.C. Translation by Benjamin Jowett.

[Pra00] Salil Pradhan. Semantic location. *Personal and Ubiquitous Computing*, 4(4):213–216, 2000. Springer-Verlag.

[Pri95] Corrado Priami. Stochastic pi-calculus. *The Computer Journal*, 38(6):578–589, 1995.

[PT00] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[Rep99] John H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999.

[RJP04] Gruia-Catalin Roman, Christine Julien, and Jamie Payton. A formal treatment of context-awareness. In *Proceedings of FASE'04*, volume 2984 of *LNCS*, pages 12–36, 2004.

[RLU94] Mike Rizzo, Peter F. Linington, and Ian Utting. Integration of location services in the open distributed office. Technical Report 14-94*, University of Kent, Computing Laboratory, University of Kent, Canterbury, UK, August 1994.

[RM02] Gruia-Catalin Roman and Peter J. McCann. A notation and logic for mobile computing. *Formal Methods in System Design*, 20(1):47–68, January 2002. ISSN: 0925-9856.

[RMP97] Gruia-Catalin Roman, Peter J. McCann, and Jerome Y. Plun. Mobile UNITY: Reasoning and specification in mobile computing. *ACM Transactions on Software Engineering Methodology*, 6(3):250–282, 1997.

[Rot03] Jörg Roth. Flexible positioning for location-based services. *IADIS International Journal on WWW/Internet*, 1(2):18–32, 2003. ISSN: 1645-7641.

[SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers & Graphics Journal, Elsevier*, 23(6):893–902, December 1999.

[SC02] Kumaresan Sanmugalingam and George Coulouris. A generic location event simulator. In Gaetano Borriello and Lars Erik Holmquist, editors, *Proceedings of the 4th international conference on Ubiquitous Computing (UbiComp)*, volume 2498 of *LNCS*, pages 308–315, Göteborg, Sweden, 2002. Springer-Verlag. ISBN: 3-540-44267-7.

[Sch95] Bill N. Schilit. *A Context-Aware System Architecture for Mobile Distributed Computing*. PhD thesis, Columbia University, May 1995.

[ST94] Bill Schilit and Marvin Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, September/October 1994. ftp://ftp.parc.xerox.com/pub/schilit/AMS.ps.Z.

[Ter06] Sotirios Terzis. Combining theory and systems building – experiences and challenges. In *International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB), Pervasive 2006*, May 7 2006.

[WBB06] Torben Weis, Christian Becker, and Alexander Brändle. Towards a programming paradigm for pervasive applications based on the ambient calculus. In *International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB), Pervasive 2006*, May 7 2006.

[Wei91] Mark Weiser. The computer for the 21st century. In *Scientific American Ubicomp Paper after Scientific American edititing*. Scientific American, 1991.

[Wei93] Mark Weiser. Hot topics – ubiquitous computing. *IEEE Computer*, 26(10):71–72, October 1993.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[WJH97] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, 1997.

[Zim05] Pascal Zimmer. A calculus for context-awareness. Report Series RS-05-27, BRICS, August 2005. ISSN: 0909-0878.