COMITY: Coordinated Application Adaptation in Multi-Platform Pervasive Systems

Verena Majuntke, Sebastian VanSyckel, Dominik Schäfer, Christian Krupitzer, Gregor Schiele*, Christian Becker

University of Mannheim
Schloss, 68161 Mannheim, Germany
{verena.majuntke, sebastian.vansyckel, christian.becker}@uni-mannheim.de
{dominik.schaefer, christian.krupitzer}@students.uni-mannheim.de

*DERI, National University of Ireland
IDA Business Park, Galway, Ireland
gregor.schiele@deri.org

***Abstract*—Pervasive applications are designed to support users in their daily lives. In order to provide their services, these applications interact with the environment, i.e. their context. They either adapt themselves as a reaction to context changes, or adapt the context via actuators according to their needs. If multiple applications are executed in the same context, interferences are likely to occur. In this paper, we present COMITY – a framework for interference management in multi-platform pervasive systems. Based on contracts specifying an application's interaction with the context, the framework automatically detects interferences and resolves them through a coordinated application adaptation. We analyze the problem of interference resolution, discuss respective algorithms and extensively evaluate our prototype.**

***Keywords*—Application Coordination; Interference Detection and Resolution; Context-aware Computing; Pervasive Environments**

I. Introduction

Intelligent applications in pervasive systems interact with their context, i.e. they are context-aware and context-altering. An important characteristic of context-aware applications is their ability to adapt to changes in their context. Adaptation can either be done by adapting the application's behavior, e.g. adjusting the brightness of a display or volume of an audio output, or by changing its structure, e.g. switching from audio output to a graphical user interface. As a consequence, applications that share the same context can influence each other. If applications are not aware that other applications are influenced by their context interaction – e.g. if an application dims the light, the video based input of an other application can degrade in quality – *interferences* can occur. Handling such interferences requires coordination of applications. In the past, we have presented an initial framework based on PCOM [2] in [17] which was able to detect interferences and to resolve them within the PCOM system using built-in adaptation mechanisms (cf. [9] and [10]).

In this paper, we present a general approach to application coordination. We extend existing systems by context contracts and an adaptation interface, enabling a thin middleware layer the coordination across different pervasive systems. The context contracts serve as a basis to detect interferences and to resolve them through a coordinated application adaptation.

The paper has three main contributions. First, we present system extensions needed for cross-platform application coordination. Secondly, we discuss the problem of interference resolution in detail and present backtracking-based algorithms that exploit typical interference characteristics. Thirdly, we evaluate the performance of the resolution algorithms and the coordination framework in general.

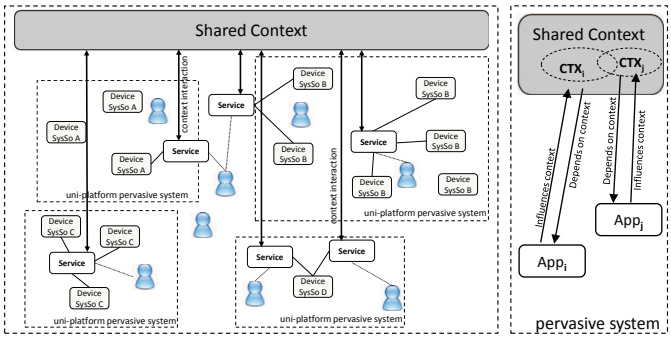
The remainder of the paper is structured as follows: At first, we describe our system model. In Section III, we briefly discuss interferences and give an overview of the coordination framework. In Section IV, we present extensions to pervasive system software that are necessary for a cross-platform coordination. We introduce our approach to interference resolution in Section V, before we discuss implementation details and evaluate our framework in Section VI. Finally, we close with a conclusion and an outlook on future work in Section VIII.

II. System Model

A *pervasive system* consists of a set of users and devices. These devices cooperate in order to provide services to the users. To realize a pervasive system, devices are equipped with respective system software, such as Aura [7], Gaia [23], and BASE/PCOM [3]/[2]. A *uni-platform pervasive system* is a pervasive system in which all devices are equipped with the same system software. Our work focuses on multi-platform pervasive systems which are illustrated in Figure 1a. A *multi-platform pervasive system* emerges if two or more uni-platform pervasive systems share the same physical space. To determine the physical space of a pervasive system the existence of a location model like [25] or [1] is assumed. The location model provides a symbolic reference for physical spaces like buildings, floors, rooms, etc.

The provision of services is realized through the execution of pervasive applications. A pervasive application is defined by three characteristics: 1) *Distribution*: A pervasive application is distributed among multiple devices. It makes use of available resources and functionalities forming the *functional configuration* of the application. 2) *Context-interactivity:* A

978-1-4673-4575-0/13/$31.00 ©2013 IEEE



(a) Multi-platform pervasive system (b) Interference

Fig. 1: System Model

pervasive application interacts with its context. On one hand, it is able to obtain context information and to incorporate the information into configuration decisions. On the other hand it has the ability to change the context, e.g. through the use of actuators. 3) *Adaptivity:* A pervasive application has the ability to adapt itself to context changes. In dynamic environments this enables the application to continue the provision of a service in different functional configurations.

For interference management we assume applications to be *cooperative*. The specifics of this cooperation are discussed in Section IV. Furthermore, we assume that each application is able to compute possible alternative functional configurations for a given context as discussed in [8].

III. Background: Interferences and Coordination Framework

In [17], we identified and discussed interferences as a major problem that is likely to occur in pervasive systems. To handle such interference at runtime, we presented a coordination framework for uni-platform pervasive systems. In this section, we give a brief introduction to the problem of interferences as well as an overview of the framework, providing basic preliminaries for the contributions of this paper. For detailed information we refer to [17].

A. Interference

The parallel execution of pervasive applications poses challenges in multi-platform pervasive systems. The problems arise from the fact that pervasive applications interact with a shared context. As a consequence, they are directly related with each other via their context and can have a significant impact on each other. Figure 1b illustrates such a situation in which two applications interact with a shared context. Consider the situation in which application App_i has changed the context according to its needs. Right after that, application App_j is started and discovers that the shared context does not satisfy its requirements. Consequently, it also adapts the context according to its needs.

The action of App_j changes the basis for the active configuration of App_i. Since its current configuration is not viable in the changed context anymore, App_i is now forced to react leaving it with two options: (1) It can adapt the context again according to it's own needs or (2) it can adapt itself. The first option may result in a cycle where the two applications take turns adapting the context. The second option may prove to be suboptimal because another configuration may not satisfy the user's requirements. Moreover, it may be possible that no viable configuration can be found at all.

The situation discussed above is a general problem in multi-user pervasive systems. Applications interact with the shared context. They make configuration decisions based on context states and adapt the context according to their needs without considering that other applications may be executed in parallel. We refer to the described problem as an interference. An *interference* is an application-induced context which forces another application to react.

B. Coordination Framework

In order to handle interferences we have developed a coordination framework as shown in Figure 2. The basic idea of the coordination framework is the realization of a middleware layer that detects and resolves interferences between applications in different uni-platform pervasive systems.

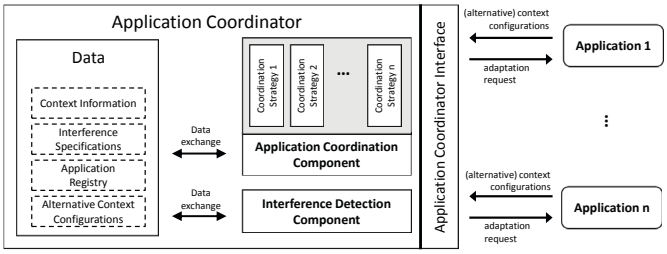


Fig. 2: Overview Application Coordinator

To ensure the management of application-specific interferences, each application is required to register at the framework and to provide *context contracts*. A context contract describes the interaction with the shared context. It depends on the functional configuration of the application. The context contract consists of the application's *interference specification* and its *context influences*. The interference specification defines the context states which pose an interference for the application. The context influences specify how the application influences the shared context. For interference detection, each application needs to provide an active context contract which depends on its active functional configuration. For interference resolution, each application needs to provide a set of alternative context contracts which depend on the application's alternative functional configurations.

Interference detection is realized by the *interference detection component*. The process of interference detection involves the evaluation of all active interference specifications for the current context. It is triggered every time the set of interference specifications or the context changes. An interference is detected if an interference specification is satisfied by the current context. In this case, a description of the interference is composed. The description includes the satisfied interference specification, the contributing context and all involved applications. Once the description is created, the interference

resolution process is triggered by invoking the *application coordination component*. Interference resolution is a two-staged process. At first, an interference resolution plan is computed according to a coordination strategy which is set for the framework. The resolution plan determines how applications have to adapt in order to solve a detected interference. The plan is computed based on the active and alternative context contracts the applications have registered. We describe the details of our interference resolution approach in Section V. Once a resolution plan has been obtained, the framework instructs applications to adapt according to the plan.

IV. System Extensions

A major challenge for a cross-system approach is the integration of applications running on various platforms into application coordination. First, irrespective of the platform, applications must be able to specify their interaction with the context and provide their adaptation options to the coordinating entity. For this, we briefly introduced the abstract concept of *context contracts* in [17]. In this paper, we discuss these contracts in detail and provide a formal description on how they are modeled. Secondly, in order to resolve detected interferences, the coordinating entity must be able to instruct applications to adapt. For this, we define a simple adaptation interface. The implementation of the contracts and the adaptation interface by an existing system realize its integration into application coordination.

A. Context Contract

A *context contract* defines the interaction of an application with its context depending on a functional configuration. For application coordination, each application needs to provide the *active context contract* for its current configuration and at least one *alternative context contracts* for alternative functional configurations. The active context contract is required for interference detection whereas the list of alternative context contracts is used for interference resolution.

A challenge for the specification of context contracts is to ensure that the shared context is addressed by all applications in a common way. This can be achieved by the use of ontologies like [5] or [13] but is not addressed in this paper. Context contracts consist of two parts, interference specification and context influences, which are discussed in detail in the following.

1) Interference Specification: The first part of a context contract is the *interference specification*. It defines the context states that pose an interference for an application.

Interference specifications have a major influence on the complexity and efficiency of the application coordination process. To select a formal model for the specifications, three aspects need to be considered: The *expressiveness* of the logic, the *efficiency of statement evaluation*, and the test on *unsatisfiability of the set of interference specifications*. We chose to base our model on monadic predicate calculus [16], as it proves to have a good balance between all aspects. With respect to expressiveness, the logic extends propositional logic by quantifiers and unary predicates. This allows to address elements in sets of objects and to make statements about them. The restriction to monadic predicates prevents the modeling of relationships which has a positive impact on the efficiency of statement evaluation. It reduces the complexity to $O(|\varphi||\mathcal{A}|)$ where $|\varphi|$ is the number or attributes in the formula φ and $|\mathcal{A}|$ is the number of possible assignments for each of these attributes. In contrast to more expressive models, such as first-order predicate logic, a check for unsatisfiability is decidable. This check is used to ensure that solutions to interferences can be found in general. The following is an example of an interference specification of a presentation application:

```
BEGIN
            temperature < 19.0 Celsius
    OR
            activity = presentation
        AND light.intensity != dimmed
    OR
            activity = presentation
        AND audio.volume > 45 decibel
END
```

The interference specification models three context states that the application encounters as an interference. The first one is a temperature that is below $19\,^{\circ}\mathrm{C}$. The second interference occurs if a presentation takes place in the environment and the lights are not dimmed. The third one models an interference when a presentation is held and the audio volume is greater than 45 decibel.

An interference specification needs to be defined by the application developer. One possibility is to automatically infer the interference specification from an application's context-action rules, i.e. the rules that describe how an application has to act in a certain context. Furthermore, user requirements towards the context can be modeled as interferences. If a user feels disturbed by loud noises while talking on the phone, the respective context state can be added to the interference specification.

2) Context Influences: The second part of the context contract are the application's *context influences*. Context influences explicitly specify the effects an application has on the shared context. They are determined through the resources and actuators that the application uses. In active context contracts, the context influences specify the actual effects on the shared context. For alternative context contracts, the actual context influences may not be known before its instantiation. An alternative context contract can state that it will affect the context with audio but it will probably not know with which intensity. Hence, applications can specify expected context influences which may cover a range of values. Once the alternative contract becomes active the explicit context influences are set. As an example, the context influences of a video presentation could be defined as follows:

```
CI = {activity = video_presentation,
      light.intensity = dimmed,
      audio.type = speech,
      audio.volume = 55 decibel}
```

The context influences state that the application sets the activity of the environment to video presentation, dims the lights and output speech with an intensity of 55 decibel.

B. Adaptation Interface

The second part of an application's cooperation is the implementation of the *adaptation interface*. The interface defines a single functionality.

```
interface Instructable {
     void adaptToCC(ContextContract cc);
}
```

The interface enables the coordination framework to instruct an application to switch into an alternative functional configuration. As a parameter, it passes the respective context contract, which has been determined to be interference-free (see Section V), to the application. The application then needs to instantiate a functional configuration that complies with this context contract.

Having discussed the integration of existing systems into application coordination, we discuss the problem of interference resolution and present our approach in the next section.

V. Interference Resolution

The process of resolving an interference can be split into two steps: (1) The computation of an interference resolution plan, and (2) the instruction of application to adapt according to the plan. The latter is achieved via a call to the adaptation interface (see Section IV-B). The former is a complex task and requires a systematic approach.

Given an interference and the set of all active and alternative context contracts, an *interference resolution plan* is a list that assigns a context contract to each active application in the pervasive system. If each application fulfills its respective assignment, the detected interference is resolved and an interference-free system state emerges. In case an application already fulfills the assigned contract no changes are required. Otherwise, it must adapt accordingly.

In order to determine the plan, the application coordination component searches for a context contract for each application such that the detected interference is resolved and no new interferences are created. That means, an assignment needs to be found for each application such that (1) the context influences of the application do not lead to the satisfaction of any active interference specification and (2) the interference specification of the application is not satisfied by the current context.

In the following, we analyze the problem of interference plan computation in detail and formalize the problem by modeling it as a constraint satisfaction problem. Subsequently, we present an example resolution algorithm which can be tailored to the specifics of an interference.

A. Interference Resolution as CSP

The computation of the interference resolution plan is a complex task. The complexity stems from the fact that context influences and interference specifications are strongly related with each other. The context influences on one hand change the context based on which interference specifications are evaluated. The interference specification on the other hand restricts the possible context influences. Changing the context contract of an application changes the context influences as well as the interference specification. Thus, using an alternative context contract may show that the contract's interference specification is satisfied by the current context as well as that the contract's context influences satisfy an existing interference specification.

In order to reason about the complexity of the interference resolution plan computation, we modeled the problem as a constraint satisfaction problem (CSP) [24] which is defined as follows:

Constraint Satisfaction Problem (CSP) A constraint satisfaction problem is a triple (V, D, C) where $V = \{V_1, \ldots, V_n\}$ is a finite set of variables and $D = \{D(V_1), \ldots, D(V_n)\}$ is a set of finite domains such that $D(V_i)$ is the finite set of potential values for V_i. Furthermore, $C = \{C_1, \ldots, C_k\}$ is a finite set of constraints where each C_l is a pair (t_l, R_l) with $t_l = (v_{l_1}, \ldots, v_{l_m})$ being an m-tuple of variables and R_l being an m-ary relation over D. A solution of an instance of a CSP is a function $f : V \rightarrow D$ such that $\forall (t_l, R_l)$ with $t_l = (v_{l_1}, \ldots, v_{l_m})(f(v_{l_1}), \ldots, f(v_{l_m})) \in R_l$.

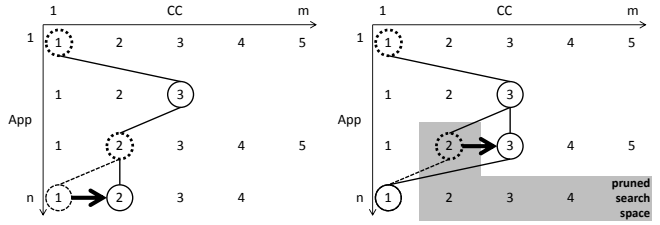
Based on the previous definition, the problem of computing an interference resolution plan can be modeled as a constraint satisfaction problem as follows:

Let V be the set of applications which are active in the environment $App = \{App_1, \ldots, App_n\}$ and let $CC(App) = \{CC(App_1), \ldots, CC(App_n)\}$ with $CC(App_i) = \{(CI_{i_1}, IS_{i_1}), \ldots, (CI_{i_m}, IS_{i_m})\}$ the finite domain of App_i namely the finite set of possible context contracts (CC) for App_i where CI are the context influences and IS is the interference specification of the contract. Furthermore, let $C = (t, R)$ be the single constraint with $t = (App_1, \ldots, App_n)$ and $R = \bigcup_{i=1}^{n} CI_{i_j} \cup CTX_{nat}(\bigcup_{i=1}^{n} IS_{i_j}) \models 0$. Thus, a solution to the problem of interference resolution plan computation is a selection of a context contract for each application such that the union of the context influences of all applications in combination with the natural context (CTX_{nat}) does not satisfy the union of all interference specifications. Please note that we have explicitly not addressed the possibility of pausing one or more applications in order to create an interference-free state if no resolution plan could be determined.

B. Resolution Algorithm

Having modeled the computation of an interference resolution plan defined as CSP, any algorithm that solves an instance of a CSP can be used to compute an interference resolution plan. Different classes of algorithms exist as discussed in [14] and [6]. Due to the dynamic nature of pervasive systems, we chose to employ a *backtracking*-based algorithm. Backtracking allows to start an immediate search, without the need to narrow the search space.

The algorithm works as follows: Given applications $App_1, \ldots, App_n$ and for each App_i at most m context contracts $CC_{i1}, \ldots, CC_{im}$, we check all $O(m^n)$ possible combinations until we find an interference-free one. Figure 3a shows an example matrix with four applications possessing between three and five contracts. The selection of one number per row depicts one combination in the matrix. The two



(a) without Pruning (b) with Pruning

Fig. 3: One Step in the Backtracking Matrix

contracts encircled with a dotted line illustrate an interference. In the algorithm, the combinations are created bottom-up, i.e. we start with the set $CC_{11}, CC_{21}, \ldots, CC_{n1}$ and shift the configuration of the lowest application possible to the right for each combination, while resetting its successors to $CC_{k1}, 1 < k \leq n$. For example, in Figure 3a, the lowest application is shifted, i.e. $k = n$.

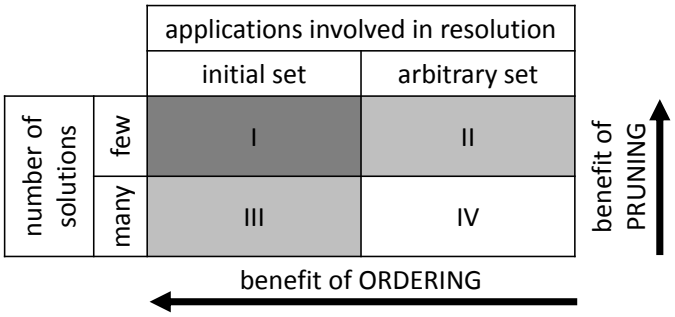


Fig. 4: Characteristics of the Resolution Problem.

An analysis of the interference resolution problem has shown that the search space is – besides the number of active applications and the size of their respective set of context contracts – characterized by two factors: (1) the number of applications which need to be adapted in addition to initially interfering applications and (2) the number of possible solutions. We exploit this fact and add two mechanisms to the classic backtracking algorithms to improve the algorithm's performance. For (1): If no additional applications need to be integrated, the applications which initially interfere are adapted at first. For this purpose, we sort the input matrix regarding the involvement of applications in ascending order (*ordering*). For (2): We apply *pruning*. Let $CC_1, \ldots, CC_i, CC_{i+1} \ldots CC_n$ be a set of active context contracts where CC_1 and CC_i cause an interference. Without loss of generality, no combination of $CC_{i+1} \ldots CC_n$ will lead to an interference-free state. Hence, the next viable combination is reached by altering CC_i and resetting $CC_{i+1} \ldots CC_n$, pruning a portion of the search space in the process. Figure 3b shows such a pruning, where no combination including CC_{11} and CC_{32} can lead to an interference-free state. The combination of the mechanisms yield four variants of the backtracking-based algorithm.

Figure 4 summarizes the anticipated effects of *ordering* and *pruning* in relation to the two main characteristics of an interference. In case no other applications than those causing the interference have to be adapted in order to resolve it, the ordering heuristic promises performance improvement in comparison to classical backtracking. With pruning, we expect the most benefit in constellations with few solutions in the search space. If in contrast further applications need to be adapted or the search space has many solutions the algorithms will perform comparable to simple backtracking.

Algorithm 1 Interference Resolution Algorithm

```
1:  procedure RESOLUTION(matrix)
2:      if ORDERING then
3:          matrix ← sortAppsByInvolvementASC(matrix)
4:      end if
5:      p ← initialCombination(matrix)
6:      while p ≠ ∅ do
7:          if isInterferenceFree(p) then
8:              return p
9:          end if
10:         p ← nextCombination(matrix, p)
11:     end while
12:     return ∅
13: end procedure
```

Algorithm 1 formulates the interference resolution approach implemented in our prototype. The input is a matrix with the set of applications as shown in Figure 3a. In the initial combination all applications use context contract at index 1. If the *ordering* mechanism is enabled the matrix is sorted in ascending order according to the involvement in an interference. To determine the ranking, we have to evaluate the set of interferences. In the worst case, each active contract interferes with every other, resulting in $\sum_{i=1}^{n} i = O(n)$ interferences. Hence, the runtime of sorting the matrix is determined by the sorting algorithm itself, i.e. it is $O(n * log(n))$. Next, the algorithm checks each combination until an interference-free state is found.

Algorithm 2 Function Determining the Next Combination

```
1:  function NEXTCOMBINATION(matrix, p)
2:      for app = n → 1 ∈ matrix do
3:          if (PRUNING ∧ isInvolved(app) ∧ app.hasNextCC) ∨
    (!PRUNING ∧ app.hasNextCC) then
4:              p.nextCCForApp(app)
5:              return p
6:          end if
7:          app.firstCCForApp(app)
8:      end for
9:      return ∅
10: end function
```

The function in Algorithm 2 determines the next combination for the resolution procedure. The function works as follows: If *pruning* is enabled, the function searches for the first application that is involved in any interference in the matrix bottom-up. As soon as it finds such an application that has further alternative context contracts, it increments its contract index and returns the new combination. In the process, it resets the contract index of every other application to 1. In case *pruning* is not active, it increments the contract index of the lowest application possible, likewise resetting those before. As mentioned earlier, with $O(m^n)$ possible combinations, the worst case runtime independent of *pruning* is $O(m^n)$.

VI. Implementation and Evaluation

In the last section, we analyzed the problem of interference resolution and discussed algorithms for their resolution. In this section, we describe the implementation of our framework COMITY, analyze memory requirements and discuss performance measurements during execution.

A. Implementation

In order to evaluate our approach in a reasonable scenario, we set up a pervasive system using our system software BASE [3]. BASE is a middleware that has been designed for pervasive systems. It has a lightweight but extensible core, which enables its operation on resource-poor devices, such as embedded systems, but also supports costly functionalities running on full-fledged devices, such as desktop computers. Devices which are equipped with BASE are able to detect each other and form a spontaneous network. In order to build and execute pervasive applications, BASE models functionalities and device capabilities as services and provides a uniform access. Each (remote) service can be accessed via local proxies implementing a defined interface. Moreover, BASE enables remote communication while shielding applications from the underlying communication technologies, interoperability protocols and communication models.

We implemented the coordination framework COMITY as a BASE service making coordination accessible to all applications in the system. While doing this, we only used BASE services that are found in most middleware-based systems, such as service discovery and remote procedure calls. Consequently, COMITY is not limited to our BASE/PCOM platform.

Next, we analyze the memory requirements of COMITY and discuss its overhead in comparison to stand-alone BASE. Afterwards, we conduct performance measurements regarding the functionality of COMITY.

B. Memory Requirements and Overhead

In order to determine the overhead caused by the coordinator in pervasive systems, we first evaluated a pure BASE setup and compared it to a COMITY setup. The footprints of both systems have been measured using the Java profiler Java VisualVM which is part of JDK. A pure BASE setup has a memory requirement of approximately 290kB. Running as a BASE service, COMITY adds up to 160 bytes. COMITY grows with the increasing number of context contracts added by applications. An average context contract with $|IS/CI| = 5$ has a size of about 340 bytes. Registered at the coordinator, an active contract adds about 1kB memory and approximately 360 bytes if alternative.

Besides the memory requirements, we also measured the messages required by BASE and COMITY. For each detection cycle, which is performed by a BASE instance every 100 ms – the time has been set to create a balance between the network traffic load and the refresh period for services – a message is sent to and returned by all other BASE instances in the system. For n BASE instances this results in $2(n-1)$ messages per instance adding up to $O(n^2)$ messages. In addition to the BASE communication, COMITY adds 2 messages for the registration process, 1 message for the addition of each alternative contract and 1 message for each adaptation request. Furthermore, we implemented leases that applications have to renew every 100 ms on the coordinator. This allows the coordinator to cope with dynamic environments and remove application information of inactive applications. This adds one extra message per application per cycle to the communication overhead.

C. Performance Measurements

Next, we evaluate the application coordination process. Figure 5 gives an overview of the entire process. It starts when an application registers for coordination at the coordinator (1). The application reports its active context contract to the coordinator, who stores it in the respective data structures (2). Once the data has been processed, the coordinator sends the registration id to the application and starts the interference detection (3). In case interferences are found, the interference resolution is triggered (4). This subprocess computes the interference resolution plan and sends the adaptation instruction to the applications (5). The registration id received in (3) can be used by the application to add alternative context contracts at any time (6). Adding alternative context contracts does not trigger the interference detection.

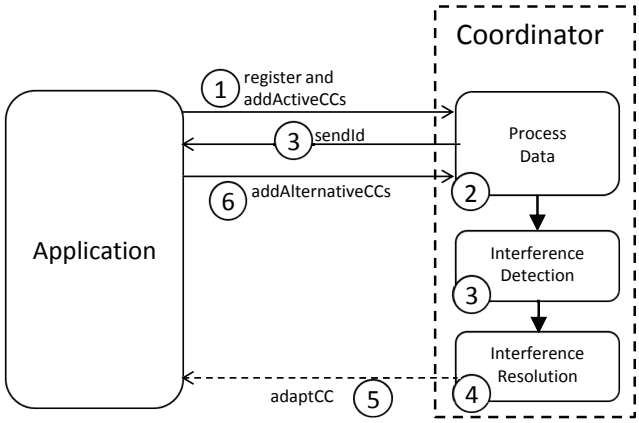


Fig. 5: Overview: Application Coordination Process

Analytically, the time required from (1) through (5) can be determined as follows:

$$T(AP) = OWNC + DP + ID + IR + OWNC$$

where $OWNC$ is the time required for a one-way network communication (1) (5), DP the time required for data processing (2), ID the time required for interference detection (3) and IR the time required for interference resolution (4). The overall effort depends on the interference detection and resolution, which are evaluated in detail in Sections VI-C1 and VI-C2. For $OWNC$, DP, and the pure registration ((1) and (2)) we obtained the values shown in Table I. Here, we altered the length of the context contracts, i.e. number of attributes in the interference specification (IS) and context influences (CI), respectively. In our test system, the the coordinator ran on a desktop PC with Intel Core2 Quad Q 6600 @ 2.40 GHz and 4 GB RAM running a 64 Bit Windows 7 and the application on

a Sony Vaio Solo U1500@1.33 Ghz with 1 GB RAM running a Windows Vista Business and being connected via LAN. For each setting, we performed a number of 20 runs and obtained the time in ms given in Table I.

Processes	$\|CI/IS\|$					
	1	4	9	16	25	36
$OWNC$	1.8	1.6	1.8	1.5	1.7	1.8
DP	1.2	1.2	1.2	1.4	1.7	1.7
$Registration$	0.5	0.5	0.45	0.45	0.45	0.45
$Overall$	5.3	4.9	5.25	4.85	5.55	5.75

TABLE I: Performance results overview

The table shows that for the size of context contracts ($|CI/IS| = \{1, 2, 4, 9, 16, 25, 36\}$), the time required for the overall process does not vary significantly. Overall, the applications required between 4.85 - 5.75 *ms* to register an active context contract at the coordinator ensuring that the interferences of the application are handled in the system.

The times in the table exclude the runtime of interference detection and resolution. We evaluated both these processes seperately on a 2x Quad-Core Intel(R) Xeon(R) @ 2.33GHz device with 6GB RAM running a 64 Bit Windows Server Standard Edition, as described in the following.

1) Interference Detection: In order to detect interferences, the process checks if any of the active interference specifications is satisfied by the current context. The brute-force approach is to evaluate every single interference specification in the set of active context contracts. We optimized this process based on the idea that only those interference specifications need to be checked, which refer to a context that has been changed recently. For example, we only check the set of interference specifications that reference the context attribute "'temperature"', if the value of "'temperature"' was changed since the last detection.

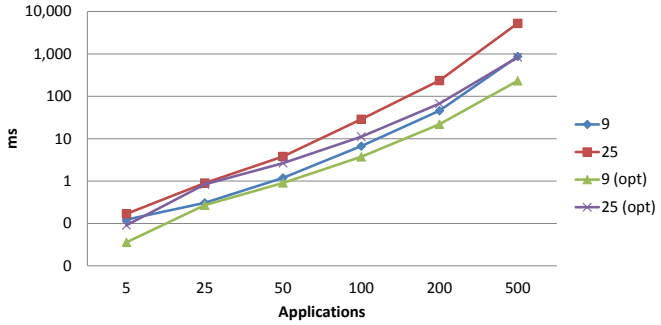
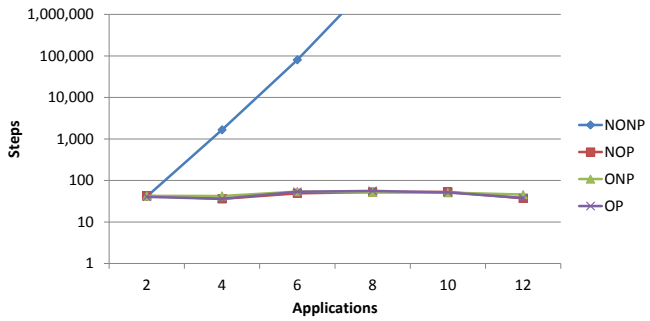


Fig. 6: Performance of the Interference Detection

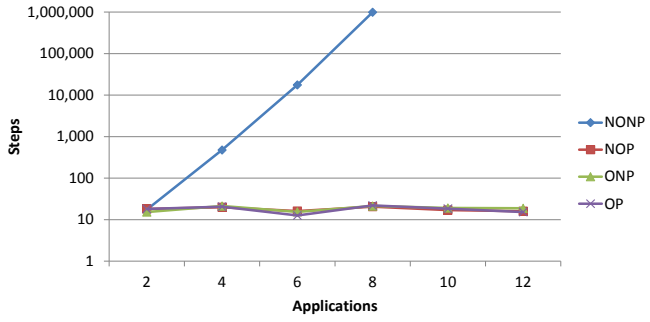
For the measurements, we set up the following scenario: We added a number of $n = \{1, 5, 25, 50, 100, 200, 500\}$ active context contracts to the coordinator that are interference-free. Afterwards, we changed the context such that an interference was created for 25% of the applications. Since the effort for interference detection depends on the number of context attributes that have to be evaluated, we used this number as the variable parameter with $|CI/IS| = \{1, 4, 9, 16, 25\}$. An extract of our measurements is shown in Figure 6. For a contract size of $|CI/IS| = 9$, for example, brute-force detection takes about $1ms$ for up to 50 applications. For 100 applications, it takes about $6.7ms$. In contrast, the optimized detection reduces the time by more than half to $3.2ms$. Given these results, we can conclude that even using the brute-force approach, the overhead for up to 50 applications – a very large amount of applications in a typical pervasive system – is acceptable in a real-time system.

2) Interference Resolution: The interference resolution process is the biggest factor in the runtime of the overall process. In Section V-B, we presented a backtracking-based algorithm with the additional mechanisms *ordering* and *pruning*. Subsequently, we have four variations of the resolution algorithm, namely NONP, NOP, ONP, and OP, where O = ordering, P = pruning, and N = the negator.

Following the characterization of interferences shown in Figure 4, we set up two test cases – test case 1 for (I) and (III), and test case 2 for (II) and (IV) – to evaluate the four variations.



(a) small solution space ($r = 1$)



(b) large solution space ($r = m/2$)

Fig. 7: Resolution Performance in Test Case 1

Test Case 1: In this test case, a solution to an interference was found by adapting those applications which were initially involved in the interference. The parameters were: (1) the number of applications $n = \{2, 4, 6, 8, 10, 12\}$, (2) the number of context contracts per application $m = \{2, 4, 8\}$, (3) the number of context contracts per application that can resolve the interference $r = \{1, m/2\}$, and (4) the number of applications involved in the initial interference with $i = \{2, n/2, n\}$. We fixed the number of attributes per context contract to $|CI/IS| = 5$.

The Figures 7a and 7b show the average number of backtracking steps required by each variation of the resolution algorithm with respect to the number of applications n and parameters $m = 8$, $r = 1$ and $r = m/2$, respectively, and $i = 2$. In test case 1, all three variations of the algorithm with either *ordering* and/or *pruning* (NOP, ONP, and OP) clearly outperform the variation NONP. In fact, NONP shows a runtime exponential in n, whereas the other three show a runtime exponential in i – the number of applications involved in the initial adaptation – or better. For variation ONP, this is due to the fact that all applications that have to be adapted are sorted to the bottom of the matrix and, therefore, altered first. For NOP and OP we defer to test case 2 for further analysis.

We conclude that variation NONP is not applicable for real-time pervasive systems, as it takes on average more than $3s$ to resolve an interference for as little as six applications in the environment, whereas the others take less than $10ms$.

Test case 2: In test case 2, the solution space is not limited, but a set of applications not involved in the initial interference have to be adapted as well. Hence, this test case is far more complex than the first one. A combination of contracts that is interference-free for the applications initially involved, may result in interferences with applications previously not involved. As a result, the search space becomes unpredictable. The parameters are the same as in test case 1 with the addition of: (5) the number of applications that need to be adapted with $a = \{n/2, n\}$.

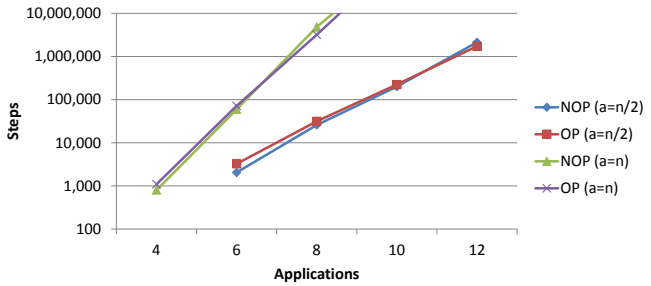


Fig. 8: Resolution Performance in Test Case 2

For test case 2, the only viable variations of the resolution algorithm are those with *pruning*, i.e. NOP and OP. As expected, the mechanism *ordering* by itself does not improve the performance, as further interferences occur during resolution. Figure 8 shows the performance of both variations NOP and OP in a test run with the parameters $m = 8$, $r = m/2$, and $a = n/2$ and $a = n$, respectively. In this very complex setting, and with $a = n/2$, the best-performing NOP takes 0.109, 1.766, 17.951, and $214.305s$ for finding a resolution in a system with six, eight, ten, and twelve applications. With $a = n$, NOP takes $3.424s$ for six, and $408.542s$ for eight applications. However, we assume that the case of having to adapt all (or near to all) applications in order to resolve an interference is rather seldom. These results show, that the runtime is exponential in a – the number of applications that need to be adapted in order to resolve the interference.

We conclude that our approach to interference resolution is, depending on a, applicable for environments of up to eight applications.

VII. Related Work

The analysis of existing work shows that subclasses of the problem of interferences have been identified and addressed under the terms interference, service interaction and conflict. Frameworks for the management of these subclasses which come close to the definition of interferences in this paper have been presented by Bortenschlager *et al.* [4] and Morla *et al.* [19]. [4] introduces the UbiCoMo infrastructure for agent coordination in pervasive systems and discusses a number of patterns to handle situations which require coordination. [19] present a general framework that allows application developers to reason about interferences offline and provides solutions to solve them. While both approaches address the problem in a general way, the considerations remain on a theoretical level.

A variety of research work exists which explicitly focuses on the problem of interference (conflict, service interaction) resolution ([15], [11], [22], [12], [26], [29], [20], [21], [31]).

The use of priorities, for example, to solve a detected conflict (interference) has widely been investigated ([15], [11], [22], [12], [28], [32], [30], [18]). Haya *et al.* [11], for example, approach the resolution of concurrent request to exclusive resources by employing preemptive priority queues. A centralized mechanism is used to store action requests on resources in queues. Each action request has a pre-defined priority. If several requests for a resource exist in a queue, the request with the highest priority is chosen. Priority-based resolution strategies are also employed for interferences (conflicts) that occur between different users. Shin *et al.* [28] dynamically assign priorities to users based on their context conflict history. If a user's context is likely to lead to a conflict according to the history, the user is assigned a low priority. Further approaches resolve conflicts based on user preferences, e.g. [27], [21], and [31]. These approaches are based on the idea that users have preferences towards services and how they are composed, e.g. use of specific resources. An interference (conflict) occurs if a service is accessed by multiple users or when multiple services share limited resources. The resolution of a detected conflict is achieved by computing service compositions trying to optimize user satisfaction based on preferences.

Further approaches propose a resolution process and combine several strategies which may also require the interaction with the user. For example, Shin *et al.* ([20], [26], [29]) developed a process consisting of three resolution strategies and which are applied depending on the characteristics of a detected conflict. The first two strategies support an automatic conflict resolution employing user preferences or user priorities. The third strategy is referred to as technology augmented social mediation. It presents a list of service recommendations to the users, takes their preference statements and computes a decision based on the input.

In contrast to our work, all of the discussed approaches do not address multi-platform systems. Interferences are assumed to occur and to be detected between pervasive applications which are run in a single system. Moreover, the discussed work focuses on specific strategies for interference resolution instead of the general management of interferences. However, these strategies can be integrated as coordination strategies into our framework for interference plan computation.

VIII. Conclusion and Future Work

In this paper we presented an approach to realize application coordination in multi-platform pervasive systems. In order for applications to be integrated, existing systems are required to implement context contracts and an adaptation interface. Based on the context contracts, interferences can automatically be detected and resolved by planning and initiating a coordinated application adaptation. For adaptation planning we discussed four algorithms, a simple backtracking algorithm and possible improvements through an ordering heuristic and pruning. Our evaluations showed that especially in cases where no further applications needed to be adapted, our heuristic significantly improved the performance of the resolution. Furthermore, we were able to conlude that the bottleneck of application coordination is the interference resolution.

The parameter that can be adapted and also has a major impact on interference resolution is the number of context contracts per application. Thus, for future work, we plan to achieve a minimal mapping of functional configurations to context contracts. Furthermore, we are working on refining our interference resolution approach with the goal of making it more suitable for interferences of type (II) and (IV).

Acknowledgment

This work was supported by the German Research Foundation (DFG).

References

[1] Bauer, M., Becker, C., Rothermel, K.: Location Models from the Perspective of Context-aware Applications and Mobile Ad-hoc Networks. *Personal and Ubiquitous Computing, Springer*, 6(5-6), pp.322-328, 2001.

[2] Becker, C., Handte, M., Schiele, G., Rothermel, K.: PCOM - A Component System for Pervasive Computing. In *Proc. Pervasive Computing and Communications (PerCom), IEEE*, 2004.

[3] Becker, C., Schiele, G., Gubbels, H., Rothermel, K.: Base - A Micro-broker-based Middleware for Pervasive Computing. In *Proc. Pervasive Computing and Communications (PerCom), IEEE*, 2003.

[4] Bortenschlager, M., Castelli, G., Rosi, A., Zambonelli, F.: A Context-sensitive Infrastructure for Coordinating Agents in Ubiquitous Environments. *Multiagent Grid Systems, IOS Press*, 5(1), pp.1-18, 2009.

[5] Chen, H., Perich, F., Finin, T., Joshi, A.: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *Proc. Mobile and Ubiquitous Systems (MobiQuitous), IEEE*, 2004.

[6] Dechter, R., Frost, D.: Backtracking Algorithms for Constraint Satisfaction Problems - A Tutorial Survey. *Information-and Computer Science Technical Report*, 1998.

[7] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project Aura: Toward Distraction-free Pervasive Computing. *Pervasive Computing, IEEE*, 1(2), pp.22-31, 2002.

[8] Geihs, K., Reichle, R., Wagner, M., Khan, M.: Modeling of Context-aware Self-adaptive Applications in Ubiquitous and Service-oriented Environments. *Software Engineering for Self-Adaptive Systems, LNCS, Springer*, 5525, pp.146-163, 2009.

[9] Handte, M., Becker, C., Rothermel, K.: Peer-based Automatic Configuration of Pervasive Applications. *International Journal of Pervasive Computing and Communications, Emerald*, 1(4), pp.251-264, 2005.

[10] Handte, M., Schiele, G., Matjuntke, V., Becker, C., Marrón, P.: 3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing. *Transactions on Autonomous and Adaptive Systems (TAAS), ACM*, 7(1), pp.10:1-10:19, 2012.

[11] Haya, P., Montoro, G., Esquivel, A., García-Herranz, M., Alamán, X.: A Mechanism for Solving Conflicts in Ambient Intelligent Environments. *Journal of Universal Computer Science*, 12(3), pp.284-296, 2006.

[12] Kolberg, M., Magill, E., Wilson, M.: Compatibility Issues Between Services Supporting Networked Appliances. *Communications Magazine, IEEE*, 41(11), pp.136-147, 2003.

[13] Korpipää, P., Mantyjarvi, J., Kela, J., Keranen, H., Malm, E.: Managing Context Information in Mobile Devices. *Pervasive Computing, IEEE*, 2(3), pp.42-51, 2003.

[14] Kumar, V.: Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine, AAAI*, 13(1), pp. 32-44, 1992.

[15] Lee, H., Park, J., Park, P., Jung, M., Shin, D.: Dynamic Conflict Detection and Resolution in a Human-centered Ubiquitous Environment. *Universal Access in Human-Computer Interaction. Ambient Interaction, Springer*, pp.132-140, 2007.

[16] Löwenheim, L.: *A Source Book in Mathematical Logic*, pp.228-251, Harvard University Press, 1879-1931.

[17] Majuntke, V., Schiele, G., Spohrer, K., Handte, M., Becker, C.: A Coordination Framework for Pervasive Applications in Multi-User Environments. In *Proc. Intelligent Environments (IE)*, 2010.

[18] Masoumzadeh, A., Amini, M., Jalili, R.: Conflict Detection and Resolution in Context-aware Authorization. In *Proc. Advanced Information Networking and Applications Workshops (AINAW), IEEE*, 2007.

[19] Morla, R., Davies, N.: A Framework for Describing Interference in Ubiquitous Computing Environments. In *Proc. Pervasive Computing and Communications Workshops (PerCom Workshops), IEEE*, 2006.

[20] Otto, F., Shin, C., Woo, W., Schmidt, A.: A User Survey on: How to Deal with Conflicts Resulting from Individual Input Devices in Context-aware Environments. In *Advances in Pervasive Computing, Adjunct Proc. Pervasive Computing (Pervasive)*, 2006.

[21] Park, I., Lee, D., Hyun, S.: A Dynamic Context-conflict Management Scheme for Group-aware Ubiquitous Computing Environments. In *Proc. Computer Software and Applications Conference (COMPSAC), IEEE*, 2005.

[22] Ranganathan, A., Campbell, R.: An Infrastructure for Context-awareness Based on First Order Logic. *Personal and Ubiquitous Computing, Springer*, 7(6), pp.353-364, 2003.

[23] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K.: A Middleware Infrastructure for Active Spaces. *Pervasive Computing, IEEE*, 1(4), pp.74-83, 2002.

[24] Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*, Pearson Education, 2003.

[25] Satoh, I.: A Location Model for Pervasive Computing Environments. In *Proc. Pervasive Computing and Communications (PerCom), IEEE*, 2005.

[26] Shin, C., Dey, A., Woo, W.: Mixed-initiative Conflict Resolution for Context-aware Applications. In *Proc. Pervasive and Ubiquitous Computing (UbiComp), ACM*, 2008

[27] Shin, C., Han, D., Woo, W.: Conflict Management for Media Services by Exploiting Service Profiles and User Preference. In *Proc. ubiPCMM*, 2005.

[28] Shin, C., Oh, Y., Woo, W.: History-based Conflict Management for Multi-users and Multi-services. In *Proc. Workshop on Context Modeling and Decision Support (Context)*, 2005.

[29] Shin, C., Woo, W.: Service Conflict Management Framework for Multi-user Inhabited Smart Home. *Journal of Universal Computer Science*, 15(12), pp.2330-2352, 2009.

[30] Syukur, E., Loke, S., Stanski, P.: Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments. In *Proc. Policy Management for Web Workshop*, 2005.

[31] Thyagaraju, G., Joshi, S., Kulkarni, U., NarasimhaMurthy, S., Yardi, A.: Conflict Resolution in Multiuser Context-aware Environments. In *Proc. Computational Intelligence for Modelling Control and Automation (CIMCA), IEEE*, 2008.

[32] Wilson, M., Kolberg, M., Magill, E.: Considering Side Effects in Service Interactions in Home Automation - An Online Approach. In *Proc. Feature Interactions in Software and Communication Systems (ICFI)*, 2007.