A Context Manager for General-purpose Operating Systems

Diogo Olsen[1], Carlos Maziero[2]
[1]Federal Institute of Paraná State (IFPR), and
Graduate Program in Computer Science, Pontifical Catholic University of Paraná State (PUCPR)
[2]Graduate Program in Applied Computing, Federal University of Technology, Paraná State (UTFPR)
Curitiba, Brazil
E-mail: diogo.olsen@ifpr.edu.br, maziero@utfpr.edu.br

***Abstract*—Context-Aware Computing aims at making computing systems able to capture context information and to adapt to it, providing context-related services and information to users. Context is any information used to characterize in some sense a person, object, or place of interest for a given user or application. Initial definitions of context-aware computing date from twenty years ago, but until now it was not yet fully made available to end users. This paper proposes a generic model of a context manager for general-purpose operating systems, alongside with its implementation and first evaluation results. This model includes methods for building, storing, and managing contexts, and an API to make such information available to applications and users.**

***Resumo*—A computação sensível ao contexto visa tornar sistemas capazes de identificar contextos e se adaptar a eles, provendo serviços ou informações relacionadas a contexto aos usuários. Contexto pode ser qualquer informação usada para caracterizar de alguma forma uma pessoa, objeto ou lugar que seja de interesse do usuário ou aplicação. A primeira definição de computação sensível ao contexto possui quase duas décadas, mas ainda hoje ela não foi completamente viabilizada para usuários finais. Este artigo propõe um modelo genérico de um gerente de contextos para sistemas operacionais de uso geral, com sua implementação e resultados de avaliação preliminares. Este modelo inclui uma forma de construir, armazenar e gerenciar contextos, além de uma API para tornar essas informações acessíveis a aplicações e usuários.**

I. Introdução

O desenvolvimento tecnológico visto nas últimas décadas tornou os dispositivos tecnológicos cada vez menores e mais baratos. Atualmente não é difícil ver pessoas usando notebooks, smartfones e tablets em qualquer lugar. Portanto é necessário que estes dispositivos consigam se adaptar para serem úteis em diversas situações.

Para que os dispositivos que usamos possam se comportar de maneira adequada, é necessário que os aparelhos acessem um conjunto de informações para adequar seu comportamento ao meio, como: luminosidade do ambiente, temperatura, hora, localização geográfica, etc. Estas informações que caracterizam o mundo a nossa volta e as situações em que estamos envolvidos podem ser chamadas de *contextos* (*C*).

Neste cenário, contexto (*C*) pode ser definido como: "qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto considerado relevante na interação entre usuário e aplicação, incluindo a própria aplicação e o usuário" [1].

Por causa da abrangência da definição de contextos diversas informações podem ser tratadas como contextos e diversas aplicações podem ser criadas/melhoradas usando contextos. Em um ambiente real, podem ser construídos um número indefinido de contextos, dependendo da necessidade do usuário/aplicação e dos recursos/sensores disponíveis.

A Computação Sensível ao Contexto (CAC - *Context-Aware Computing*) foi definida pela primeira vez da década de 1990 [2] e sua idéia básica é simples: um sistema deve ser capaz de detectar o contexto em que está inserido e, com base nesse contexto, o sistema deve se adaptar para melhorar a interação com o usuário. Porém, após diversas pesquisas, a CAC mostrou-se difícil de ser implementada [3].

Este trabalho define um modelo para um gerente para contextos (*GC*) que: dilui a complexidade de trabalhar com contextos ao dividir o processo de construção em etapas; facilita a construção e distribuição de contextos ao centralizar este processo em um gerente; facilita o uso de contextos pelas aplicações, ao oferecer contextos prontos para serem usados; armazena séries históricas destes dados, para uso posterior. Não faz parte do escopo deste trabalho definir os algoritmos usados para construir contextos.

O restante deste documento está dividido da seguinte forma: a Seção II apresenta o modelo de dados proposto para representar contextos; a Seção III descreve o ciclo de vida da informação da captura até que se torne um contexto e apresenta um exemplo de como o modelo de dados pode ser aplicado para construir contextos; a Seção IV descreve o gerente de contextos; a Seção V apresenta o protótipo implementado e conclui o trabalho.

II. Modelo Proposto

"Contexto pode ser visto como um conjunto de estados ambientais, que podem ser usados para determinar o comportamento de uma aplicação ou ocorrência de um evento de uma aplicação que é de interesse do usuário ou do sistema" [4]. Com base nesta definição, um contexto pode ser visto como um conjunto de valores, aqui chamados de *informações contextuais* (*ic*). Estas informações contextuais (*ic*) devem estar relacionadas entre si, de forma que caracterizem a situação de uma entidade e devem estar construídas em um

978-0-7695-4929-3/12 $26.00 © 2012 IEEE
DOI 10.1109/SBESC.2012.38

nível de abstração adequado para serem usadas por outros aplicativos/sistemas. Formalmente, temos:

$$C = \{ic_1, ic_2, ic_3, \ldots, ic_n\}$$

onde:

- C: Contexto
- ic: Informação Contextual

Por sua vez uma informação contextual é constituida por uma série histórica de valores, ou seja, uma sequência de todos os valores que aquela informação assumiu em um determinado intervalo de tempo; cada valor da série está associado a um instante de tempo no qual este valor foi considerado válido. Essa definição de informação contextual pode ser apresentada formalmente como:

$$ic = \{< v_1, t_1 >, < v_2, t_2 >, \ldots, < v_n, t_n >\}$$

com:

$$t_0 < t_1 < \ldots < t_n$$
$$t_i - t_{i-1} = \Delta_t$$
$$n, i \in \mathbb{N}$$

onde:

- ic: Informação contextual
- v: Valor que a ic assumiu em determinado instante de tempo
- t: Instante de tempo
- Δ_t: Intervalo de tempo
- t_n: Instante de tempo atual
- n: Número de valores armazenados
- $< v,t >$: Par formado pelo valor (v) da informação contextual (ic) e o instante de tempo (t) onde este valor foi considerado válido.

Para determinar o intervalo de tempo (Δ_t) onde determinado par ($< v_n, t_n >$) foi considerado válido, é necessário determinar o intervalo de tempo Δ_t entre o par $< v_n, t_n >$ e o próximo par da sequência $< v_{n+1}, t_{n+1} >$. Desta forma, determinado valor (v) foi considerado válido no intervalo de tempo: $\Delta_t = [t_n, t_{n+1})$. A partir deste momento, o valor considerado válido é v_{n+1}.

Vale ressaltar, que dentre todas as tuplas armazenadas na série histórica, o último valor da série, com o instante de tempo t mais próximo do instante de tempo atual, é chamado de *valor corrente*, enquanto os outros valores são chamados de *valor passado*. Portanto, valores correntes e passados podem ser definidos por:

- **Valor Corrente** (v_c)**:** Valor associado a determinada informação contextual que é considerado válido para o instante de tempo atual;
- **Valor Passado** (v_p)**:** Valor associado a determinada informação contextual e a um determinado instante de tempo no qual o mesmo foi considerado um valor corrente (v_c).

Com o mapeamento de parte do mundo real para dentro do mundo virtual, cada característica de uma ou mais entidades são representadas pelas informações contextuais (ic), as quais são agrupadas em contextos (C). Neste modelo, uma informação contextual pode estar presente em diversos contextos.

III. Construção de Contextos

Para que contextos possam ser construídos, é necessário que as informações que serão processadas para construí-lo sejam adquiridas. Este processo pode ser realizado por: sensores, como no caso de temperatura e luminosidade; por software, como no caso de hora e recursos computacionais; ou por entrada de dados realizada pelo usuário.

Os dados iniciais que serão usados para construir contextos podem ser chamados de *entidades reais* (er), e são as entidades do mundo real que serão mapeadas para dentro do mundo computacional. Após a leitura destes valores, eles são armazenados em *atributos* (at) e estes atributos podem ser organizados em *entidades* (E), as quais podem ser consideradas abstrações do mundo real. Atributos e entidades podem ser definidos por:

$$at = \{< v_1, t_1 >, < v_2, t_2 >, \ldots, < v_n, t_n >\}$$

com:

$$t_0 < t_1 < \ldots < t_n$$
$$t_i - t_{i-1} = \Delta_t$$
$$n, i \in \mathbb{N}$$

onde:

- at: Atributo
- v: Valor que o at assumiu em determinado instante de tempo
- t: Instante de tempo
- Δ_t: Intervalo de tempo
- t_n: Instante de tempo atual
- n: Número de valores armazenados
- $< v,t >$: Par formado pelo valor (v) do atributo (at) e o instante de tempo (t) onde este valor foi considerado válido.

$$E = \{at_1, at_2, at_3, \ldots, at_n\}$$

onde:

- E: Entidade
- at: Atributo

Um atributo pode não estar associado a nenhuma entidade ou a várias. Atributos podem ser organizados em um conjunto de todos os atributos AT, assim cada entidade E é um subconjunto deste conjuto AT, ou seja, $E_j \subset AT \ \forall j$.

Após o processo de mapeamento das entidades reais em atributos, ocorre o processamento destes dados para a construção dos contextos. Os dados contidos nos atributos são considerados dados de baixo nível, com um nível de abstração inadequado para ser usado como um contexto. Esta é a principal diferença entre atributos e informações contextuais. Esta diferença de nível de abstração é o processo que deve ser realizado com os atributos(at), para que estes possam ser transformados em informações contextuais (ic)

Da mesma forma que atributos at podem ser organizados no conjunto de atributos AT, os contextos C podem ser

organizados em um conjunto de todos os contextos e informações contextuais CT. Desta forma cada contexto C é um subconjunto do conjunto CT, logo $C_j \subset CT \ \forall j$.

O primeiro estágio da construção de contextos é processar uma entidade real (*er*) para obter um atributo (*at*). Este processo é realizado por funções denominadas *funções coletoras* ($fc()$).

Uma função coletora ($fc()$) é uma função responsável pelo mapeamento de entidades reais (*er*), usando software ou hardware, para dentro do mundo computacional, mapeando-as em atributos (*at*). Cada dado coletado é um valor corrente (v_c) de algum atributo (*at*). Uma função coletora também pode obter dados de outros atributos ou de informações contextuais. Uma função coletora pode ser definida por:

$$at = fc(er_1, er_2, er_3, \ldots)$$

onde:

- *at*: Atributo.
- fc: Função coletora.
- *er*: Entidades do mundo real que serão mapeadas no atributo at_x.

As funções coletoras são ativadas seguindo uma periodicidade pré-estabelecida de leitura dos valores dos atributos (*at*), ou quando ocorre alguma mudança no mundo real de interesse do atributo que o mapeia. Quando alguma mudança foi percebida por sensores ou pelo sistema, a função coletora ($fc()$) é ativada para atualizar o valor do atributo (*at*).

Funções coletoras serão normalmente funções que recebem dados do próprio sistema operacional, funções baseadas em visão computacional, funções que realizam a leitura de sensores, entre outras.

As funções responsáveis por processar atributos e criar informações contextuais são as *funções agregadoras* ($fa()$). Estas funções devem usar os atributos para construir informações contextuais em um nível de abstração adequado para ser usado pelos processos. Funções agregadoras podem ser definidas por:

$$ic = fa(at_1, at_2, at_3, \ldots)$$

onde:

- *ic*: Informação contextual.
- fa: Função agregadora.
- *at*: Atributos do conjunto de entidades ET.

As funções agregadoras tendem a ser funções baseadas em inteligência artificial, funções heurísticas, funções de tomada de decisões, algoritmos de reconhecimento de padrões, mineração de dados, ontologias, entre outros tipos de funções.

Quando um contexto estiver pronto para ser entregue aos processos interessados, o *mecanismo de entrega* (*me*) realiza a entrega dos contextos aos processos. Diversos mecanismos de comunicação entre processos podem ser usados para fornecer estes dados aos processos interessados, como: RPC (*Remote Procedure Calls* - Chamada Remota de Procedimentos), DBus, canais de eventos, Corba, Sockets, Pipes, filas de mensagens, memória compartilhada entre outros.

A figura 1, apresenta uma visão geral sobre o modelo de dados usado na construção de contextos. Nesta figura estão representadas as entidades reais (*er*), funções coletoras ($fc()$), o conjunto de atributos (AT) constituído de suas entidades (E) e atributos (*at*), funções agragadoras ($fa()$), conjunto de contextos (CT) que é composto de contextos (C) e informações contextuais (*ic*), mecanismo de entrega (*me*) e processos (p) interessados nos contextos.

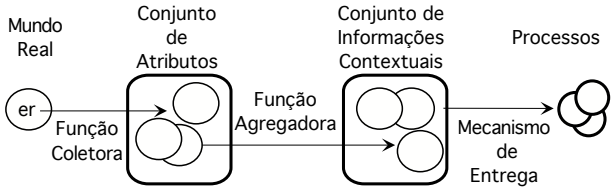

Figura 1. Visão geral do modelo de dados.

IV. Gerente de Contextos

O *gerente de contextos* (*GC*) é uma camada de software, responsável por construir contextos e entregar estes contextos às aplicações interessadas. Ao oferecer um gerente de contextos como um serviço dentro do sistema operacional, evita-se o retrabalho quando dois processos precisarem do mesmo contexto e facilita-se o acesso aos contextos, pois estes estarão centralizados em uma única *API* de acesso.

Ao realizar o processo de construção de contextos em duas etapas, uma de aquisição e outra de processamento, é possível que um dado coletado possa ser usado em diversos contextos. Ao coletar, apenas uma vez as redes sem fio disponíveis é possível gerar vários contextos como: recursos computacionais disponíveis, localização do usuário, contextos sobre nível de segurança, presença de determinado dispositivo, etc.

Caso este processo fosse realizado de uma forma descentralizada, cada contexto gerado e baseado nas redes disponíveis precisaria adquirir o mesmo dado, consumindo recursos computacionais desnecessários. Ao centralizar o processo em um gerente e permitir que atributos sejam compartilhados, o tempo de processamento do gerente possui um aumento linear, ou menor, de processamento conforme aumenta o número de contextos.

O (*GC*) é uma camada de software capaz de construir contextos, segundo o modelo apresentado na seção III, e pode ser visto na figura 2. Os componentes do gerente são: conjunto de funções coletoras (FC) e suas funções coletoras ($fc()$); conjunto das entidades (ET) suas entidades (E) e atributos (*at*); conjunto de funções agregadoras (FA) e suas funções agregadoras ($fa()$); conjunto de contextos (CT) seus contextos (C) e informações contextuais (*ic*); escalonador (*es*); e API de aceso aos dados.

O escalonador (*es*) é o mecanismo responsável por executar as funções FC, FA e o ME, para que estas possam obter os valores dos atributos (*at*), com estes construir os contextos (C) e entregá-los. O escalonador funciona executando determinada função e posteriormente agendando sua próxima execução para

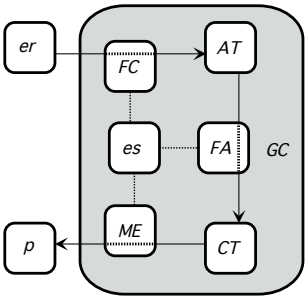

Figura 2. Visão geral do gerente de contextos.

um próximo intervalo de tempo pré-definido. Além disso, o escalonador também deve tratar a ocorrência de casos nos quais um contexto seja solicitado, porém ainda não tenha sido gerado. Neste caso, o escalonador deve executar as funções coletoras necessárias para obter os atributos e então executar as funções agregadoras para gerar o contexto solicitado e entregá-lo ao processo interessado.

Outra situação que deve ser prevista pelo escalonador é o caso de alguma mudança ocorrer no estado de algum atributo. Neste caso o sensor/software responsável por sentir este atributo deve avisar o escalonador, para que a função coletora associada possa ser executada e atualize o valor do atributo. Assim é possível deixar cada função coletora ou agregadora para ser executada em intervalos de tempo longos, e quando um contexto for solicitado ou quando alguma mudança ocorrer na entidade real o contexto é gerado, economizando recursos computacionais.

O *GC* usa uma *API* que permita que usuários e processos configurem o sistema, esta *API* possui as seguintes funcionalidades: cadastrar/descadastrar um atributo/informação contextual; associar/desassociar um atributo de uma entidade ou uma informação contextual de um contexto; ativar/desativar a construção de um atributo ou informação contetual; informar valor de um contexto, informação contextual ou atributo; listar o conjunto de informações disponíveis.

V. Implementação e Considerações Finais

Para experimentar este trabalho, foi desenvolvido um protótipo na linguagem *Python* (versão 2.7). A implementação do GC usa duas tabelas: uma para armazenar o conjunto de entidades (ET), seus atributos (at) e funções coletoras ($fc()$); e outra tabela com o conjunto de contextos (CT), suas informações contextuais (ic) e suas funções agregadoras ($fa()$). Além disso o GC possui o escalonador responsável por executar o gerente.

Para experimentar o funcionamento do sistema foi usado o contexto conectividade que verifica quais interfaces de rede estão ou podem ser conectadas e o contexto assume um dos seguintes estados: “cabeada”, “sem fio”, “celular”ou “sem conectividade”. Foi desenvolvido um cliente para o GC, este cliente foi chamado de *Gerente de Redes* (*GR*). Os testes foram realizados em um computador com Linux que tem as interfaces de rede celular, sem fio e cabeada.

Durante a execução do experimento as redes foram sendo ativadas e desativadas manualmente de forma que o contexto ($C_{conectividade}$) mudasse de valor. O contexto ($C_{conectividade}$) e os valores dos atributos (at) e informações contextuais (ic) foram inspecionados para verificar se estes assumiriam os valores esperados. Durante o experimento, todos as variáveis assumiram os valores esperados.

Quanto ao desempenho do GC, ele dependerá do número de atributos e informações contextuais ativados, da forma como suas funções coletoras e agregadoras obtêm seus valores e da periodicidade com que estas funções são executadas. Quanto mais execuções ocorrerem, mais pesado ficará o GC.

Existem trabalhos que desenvolveram formas de adquirir e tratar contextos, como [5] que define um framework que se baseia principalmente na localização do usuário/dispositivo para construir contextos e usa uma rede de sensores de presença para este fim. Apesar de possuir uma abordagem parecida com este trabalho, o framework não possui foco em reusar dados e não se preocupa em permitir que outras aplicações sejam criadas com sensibilidade a contextos.

Atualmente outros sistemas são criados para gerenciar casas inteligentes [8], para tornar as construções mais seguras [6] e para auxiliar pessoas com comprometimentos cognitivos [9]. Estes sistemas poderiam se apoiar no gerente de contextos descrito neste trabalho para construir suas aplicações

Referências

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *HUC ’99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK: Springer-Verlag, 1999, pp. 304–307.

[2] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *WMCSA ’94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1994, pp. 85–90.

[3] G. D. Abowd and E. D. Mynatt, “Charting past, present, and future research in ubiquitous computing,” *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, pp. 29–58, 2000.

[4] G. Chen and D. Kotz, “A survey of context-aware mobile computing research,” Hanover, NH, USA, Tech. Rep., 2000.

[5] Y. A. Ahn and J. S. Park, “Spatio-temporal context manager in an open context awareness framework,” *Networked Computing and Advanced Information Management*, vol. 2, pp. 681–684, 2008.

[6] D. M. Beder and R. B. de Araujo, “Towards the definition of a context-aware exception handling mechanism.” Los Alamitos, CA, USA: IEEE Computer Society, 2011, pp. 25–28.

[7] S. Schefer-Wenzl and M. Strembeck, “Modeling context-aware RBAC models for business processes in ubiquitous computing environments,” in *Proc. of the 3rd International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC)*. Vancouver, Canada: IEEE, 2012.

[8] A. Hristova, A. M. Bernardos, and J. R. Casar, “Context-aware services for ambient assisted living: A case-study,” in *Applied Sciences on Biomedical and Communication Technologies, 2008. ISABEL ’08. First International Symposium on*, 2008, pp. 1–5.

[9] B. Das, A. M. Seelye, B. L. Thomas, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, “Using smart phones for context-aware prompting in smart environments.” in *Consumer Communications and Networking Conference (CCNC)*. IEEE, 2012, pp. 399–403.